Formal Methods for Modelling Wireless Sensor Networks

Ph.D. Thesis

Csaba Biró

Eötvös Loránd University

Ph.D. School in Computer Science

Director: **Dr. Erzsébet Csuhaj–Varjú**

Ph.D. Program in Foundations and Methodology of Informatics

Director: **Dr. Zoltán Horváth**

Supervisors:

Dr. Tamás Kozsik

Dr. Gábor Kusper

Budapest

2019

Contents

1 Introduction

In today's Internet of Things (IoT) applications, Wireless Sensor Networks ($WSNs$) provide a rapid and flexible solution for accessing information in many real-world applications. A WSN deploys a large number of small, inexpensive, self-powered devices that can sense their environment and gather local information used to make global decisions. Although the research on $WSNs$ was originally motivated by military applications, other applications have also become feasible as the corresponding technology is getting more and more accessible by the public. Such applications [40] are, for instance, (1) *infrastructure security* to provide early detection of any potential threat in critical facilities such as power plants, oil refineries, airports, etc., (2) *environmental and habitat monitoring* to study the response of vegetation and fauna to climatic changes, (3) *traffic monitoring* to control traffic lights and to plan alternatives routes in order to avoid traffic jams and accidents. The sensors - that can be considered the nodes of network - are capable of processing a limited amount of information and furthermore wireless communication. The placement of sensors can be either deterministic or random, but the final location depends largely on the attributes of the area or building, or simply on accessibility. While deterministic deployment takes into consideration the environmental parameters and the field of application to place the sensors strategically, the random method of deployment makes the sensors final position unpredictable. Another important trait of such networks is about the type of nodes it contains. We can differentiate homogeneous and heterogeneous nodes that made up a network. The most critical aspect of a wireless sensor network is its fault tolerance. Many challenges are to be faced to ensure the appropriate level of fault tolerance, like problems with the power supply, hardware failures, communication errors/ disruptions, malicious attacks. Nowadays the most intensively researched field deals with the optimization of the sensor deployment locations to ensure more effective power-consumption and communication, and an acceptable level of fault tolerance.

During the research we examined the theoretical and technological solutions to the current problems and challenges of the area. Our objectives have been formulated in the light of all of this.

Before describing the objectives and the theses we would like to introduce some of the concepts that are important for their understanding.

- Propositional satisfiability (SAT) is the problem of determining, for a formula of the propositional logic, if there is an assignment of truth values to its variables for

which that formula evaluates to true. By SAT we mean the problem of propositional satisfiability for formulas in conjunctive normal form (CNF).

- Satisfiability Modulo Theories (SMT) is the decision problem of checking satisfiability of logical formulas with respect to some background theory. The most common examples for theories are the integer numbers, the real numbers, the fixed-size bit-vectors, and the arrays. The logics that one could use might differ from each other in the linearity or non-linearity of arithmetic, the presence or absence of quantifiers, or in the presence or absence of uninterpreted functions. Many problems of interest, which can be encoded as SMT problems, may require also to find models that are optimal with respect to some objective function.

- Optimization Modulo Theories (OMT) is an extension of SMT with objective functions to maximize or minimize. An OMT problem, therefore, contains not only a formula to satisfy, but also an expression $\max : obj$ or $\min : obj$, where obj denotes the objective function.

Objectives and Theses

This section summarizes the major contributions and the impact of the work by the author. According to the traditions of PhD dissertations the editorial "we" is used for describing the details of main results.

Objectives

O. I We used zeroth-order logic which gives a limited set of tools to examine this field of application:

- We have examined whether we can create a SAT-based representation to describe a randomly deployed wireless sensor network consisting of heterogeneous nodes.
- We have also examined whether SAT solvers can effectively solve this representation and if so what kind of solver should be used including sequential and also parallel solving methods.

O. II We used first-order logic which gives a richer set of tools to examine this field of application:

- We have examined whether we can create an SMT based representation of a randomly deployed wireless sensor network consisting of heterogeneous nodes that takes into account physical parameters of a real-world sensor enabling network lifetime investigation.

- We also have examined whether *OMT* solvers can optimize the lifetime of networks represented this way and if so what kind of *OMT* solver should be used.

O. III We used graph theory and their tools to examine this field of application:

- We have examined whether we can create local density and redundancy based metrics for the k-hop environment of nodes other than the known ones.
- We have also examined whether we can apply these metrics to be used for more sophisticated ranking and classification of nodes than metrics known so far.

Theses

T. I We defined a new 2-*SAT* class, the *Black-and-White 2-SAT* problem. We proved that all strongly connected graphs can be represented as a *Black-and-White 2-SAT* problem. This representation can be used generally to represent any *WSNs*. We have created a problem-specific *SAT* solver, called `BaW 1.0`, that solves these problems in linear time.

Related publications to thesis I:

- **Cs. Biró** and G. Kusper and T. Tajti, *How to generate weakly nondecisive SAT instances* 2013 IEEE 11th International Symposium on Intelligent Systems and Informatics (SISY), pp. 265-269, 2013.
- **Cs. Biró**, G. Kovásznai, A. Biere, G. Kusper, G. Geda, *Cube-and-Conquer approach for SAT solving on grids*, Annales Mathematicae et Informaticae 42 pp. 9-21, 2013.
- **Cs. Biró**, G. Kusper, T. Radványi, S. Király, P. Szigetváry, P. Takács, SAT Representation of Randomly Deployed Wireless Sensor Networks, *Proceedings of the 9th International Conference on Applied Informatics*, pp. 101-111, 2014.
- G. Kusper and **Cs. Biró**, Solving SAT by an Iterative Version of the Inclusion-Exclusion Principle, *SYNASC 2015*, IEEE Computer Society Press, pp. 189-190, 2015.
- G. Kusper, **Cs. Biró**, Gy. B. Iszály, *SAT solving by CSFLOC, the next generation of full-length clause counting algorithms*, Future IoT Technologies (Future IoT), 2018 IEEE International Conference, pp. 1-9, 2018.
- **Cs. Biró**, G. Kusper, *Equivalence of Strongly Connected Graphs and Black-and-White 2-SAT Problems* Miskolc Mathematical Notes, Vol. 19, No. 2, pp. 755-768, 2018.

- **Cs. Bíró**, G. Kusper, *BaW 1.0 - A Problem Specific SAT Solver for Effective Strong Connectivity Testing in Sparse Directed Graphs*, IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI 2018), pp. 160-165, 2018.

T. II We have created an *SMT* based representation of a randomly deployed wireless sensor network consisting of heterogeneous nodes that relies on *RF* energy model and which can be optimized in the lifetime of the network. We have defined a novel model-based incremental optimization process which is based on the results of an *SMT* solver. We have developed a problem-specific *OMT* solver that is order of magnitude more efficient than state-of-the-art solvers in case of monotonous *OMT* problems.

Related publications to thesis II:

- **Cs. Bíró** *Botond - a Simulation and Optimization Framework for Wireless Sensor Networks*, 1st International Conference on Future RFID Technologies and host the Workshop on Smart Applications for Smart Cities Eger, Hungary 6-7 November, conference talk, 2014.
- G. Kovásznai, **Cs. Bíró**, B. Erdélyi *Generating Optimal Scheduling for Wireless Sensor Networks by Using Optimization Modulo Theories Solvers* CEUR Workshop Proceedings Vol-1889, 15th International Workshop on Satisfiability Modulo Theories - SMT 2017, pp. 15-27, 2017.
- G. Kovásznai, **Cs. Bíró**, B. Erdélyi *Puli - A Problem-Specific OMT Solver*, 16th International Workshop on Satisfiability Modulo Theories - SMT 2018, Paper: 362, 10 p., 2018.
- G. Kovásznai, B. Erdélyi, **Cs. Bíró** *Investigations of graph properties in terms of wireless sensor network optimization*, IEEE International Conference on Future IoT Technologies, Future IoT 2018, IEEE, pp. 1-8. 2018.

T. III We have defined three novel density- and three novel redundancy-based local metrics. We have compared these new metrics to known ones and we have shown how they can be used for ranking and classifying nodes.

Related publications to thesis III:

- **Cs. Bíró** *Botond - a Simulation and Optimization Framework for Wireless Sensor Networks*, 1st International Conference on Future RFID Technologies and host the Workshop on Smart Applications for Smart Cities Eger, Hungary 6-7 November, conference talk, 2014.
- **Cs. Bíró**, G. Kusper *Some k-hop Based Graph Metrics and Node Ranking in Wireless Sensor Networks*, Annales Mathematicae et Informaticae, Accepted manuscript doi: 10.33039/ami.2018.09.002

Organization

In the dissertation, we also present new results that are not closely related to the individual theses, but they were important stations on the road to achieve these results. In the following, the dissertation is divided into 4 major parts. Before these parts in Chapter 2, we give an overview of the research area and the theoretical background which are needed to understand the results presented in this dissertation. In Part I, we present two zeroth-order logic-based models. These are in another approach actually, can be considered which as *SAT* instance generators. We show how to generate hard combinatorial *SAT* instances (*WnDGen*) and present a propositional logic formula (*Black-and-White 2 SAT*) to model a directed graph.

In Part II first, we present two sequential *SAT* solvers (`CSFLOC`, `BaW 1.0`), and show how to use them to solve *WSN*-based *SAT* problems. At the end of this part, we show how to develop techniques for using distributed computing resources to efficiently solve instances of the propositional satisfiability problem.

In Part III first, we introduce an *OMT* formalization of the aforementioned optimization problem for *WSNs*, provide a single-hop *WSN* simulation environment with one of the most common wireless sensor node types, propose several *OMT* benchmarks extracted from the *WSN* simulation. Also in this part, we show how to integrate this idea in search algorithms in the *OMT* framework and introduce a novel *OMT* solver called `Puli`.

In Part IV, we introduce several novel k-hop based density and redundancy metrics: Weighted Communication Graph Density ($\mathcal{WCGD}$), Relative Communication Graph Density ($\mathcal{RCGD}$), Weighted Relative Communication Graph Density ($\mathcal{WRCGD}$), Communication Graph Redundancy ($\mathcal{CGR}$) , Weighted Communication Graph Redundancy ($\mathcal{WCGR}$). We compare them to known graph metrics, and show that they can be used for node ranking. Finally, the last chapter concludes and summarizes the major results of the dissertation.

2 Background and Related Work

In this chapter, we give an overview of the research area, the theoretical background and the related work which are needed to understand the results presented in this dissertation.

2.1 Introduction

The modelling and analysis of complex networks is an important interdisciplinary field of science. The networks belong to the field of graph theory. It is known that topology represents the properties of the whole network structure. A topology describes a real network *(with constraints)* and it can be converted to an undirected or directed graph. The common property of topological models is that they are usually calculated based on probabilities [15, 9, 27, 110]. Topological metrics commonly used on networks: number of nodes and edges, average degree, degree distribution, connectedness, diameter, number of independent paths. The objects of the model can be matched by the vertices of the graph. Edges can be used to describe the relations between the objects. Graph-based modelling can be one of two types: ad-hoc or measurement-based. On large wireless networks the traditional measurements based procedures [29, 30, 62, 63, 64, 122, 184, 185, 179] can not be applied efficiently, but k-hop based approaches can be computed effectively also for large networks. There are many graph-based metrics for modelling complex networks [70].

In addition to graph theory, there are other tools for modelling and analyzing networks. The increasing complexity of networks requires other formalism [46] *(Propositional Logic, First-Order Logic, Higher-Order Logic, Temporal Logic, Intuitionist Logic, Hoare Logic, etc.)* with more advanced techniques *(Proof Assistants* [61, 77, 108], *Automated Theorem Provers* [59, 98, 146], *Model Checking tools*[21, 37], *etc.)* for analysis. These formalism and techniques are an important part of formal methods.

In Theoretical Computer Science formal methods are those techniques and tools which are used for formal specifications, proofs, model checking, and abstraction of complex systems [88, 89, 95, 109].

Within the framework of the project *(European Union and the State of Hungary, co-financed by the European Social Fund in the framework of the TAMOP-4.2.2.C-11/1/ KONV-20120014)* we started to deal with *WSNs* and *RFID* systems. *WSNs* are a special and important agent for networks. On the one hand, the *WSNs* serve as the basis for today's *IoT* applications, on the other hand, it has many limitations *(low battery life, redundant data acquisition, low duty cycle, etc.)* [106, 85].

Parameters for measuring the effectiveness of *WSNs*: scope and coverage, scalability, expected transmission number, hop count *(number of hops)*, power consumption / lifetime. In this dissertation, we also present our results in interpretation on this type of networks, but we note, that most of the results that we have presented can be applied on a wide range of networks as well. From the point of view of formal methods, during the research we used several tools for network representation and analysis. Therefore, in the rest of the thesis we will show also propositional, first-order logic and graph-based methods and tools.

2.2 Wireless Sensor Networks

A *WSN* consists of a number of spatially distributed sensor nodes, which cooperatively monitor physical or environmental conditions. They have the advantage that they consist of sensors with low energy consumption, which can be deployed easily in a cheap way on such areas which are out-of-the-way. There are three main components in a *WSN*. These are the sensor nodes, sink and monitored events. The sensors are the nodes of *WSN*. They are capable of processing some limited information and using wireless communication [3, 12, 28, 148]. A sink/base station (BS) is the type of sensor node which possesses high power, large memory and it is the entity, where information is required.

In today's IoT applications provide a rapid and flexible solution for accessing information in many real-world applications [80]. Two technologies were traditionally considered as the key enablers for the IoT paradigm: the *RFID*, and the *WSN* [106, 85].

Daily activities of our society are running by applications of the IoT, ranging from traffic monitoring devices to vehicles and home appliances. Only in 2017, the number of IoT devices has increased 31% to 8.4 billion and it is also estimated that there will be 30 billion IoT devices by 2020 [144]. Ensuring safety, security and dependency of IoT devices is therefore critical to guarantee that our software-driven world is functioning as expected. Data transmission and exchange among IoT devices is materialized using *WSN*s. Reliability of *WSN*s is thus imperative for IoT [151].

Depending on the application, different architectures [6, 155] and design modelling tools with goals/constraints [60] have been considered for *WSN*s.

Sensing models The sensors generally have different theoretical models and physical characteristics. Generally true that depending on the type of sensors the detection probability of a sensor may decay with distance, environmental conditions, and hardware configuration [41, 130, 197]. There are two general sensing models: the binary disc *(boolean)* and the probabilistic. The binary disc model is the simplest model, in the sense of this model a node is capable of sensing only from points that lie within its sensing range *(r-sensing radius*) and not from any point beyond it. In binary disc model where the detection probability p is typically calculated using the Euclidean distance where

$$dist(t_i,d_j) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2)} \tag{2.2.1}$$

is the distance between the deployment d_j point and target location t_i.

Detection probability in the binary disc model is

$$p(a_{ij}) = \begin{cases} 1 & if \quad dist(t_i,d_i) \leqslant r \\ 0 & otherwise \end{cases} \tag{2.2.2}$$

Although the binary detection model simplifies the analysis, it may not be realistic in many cases. The probabilistic sensing model is a more actual perception, which can be taken as an extension of the binary disc sensing model.

Detection probability in the probabilistic sensing model [182] is

$$p(a_{ij}) = \begin{cases} e^{-\beta dist(t_i,d_j)} & if \quad dist(t_i,d_j) \leqslant r \\ 0 & otherwise \end{cases} \tag{2.2.3}$$

where β [195] is a parameter related to the physical characteristics of the sensor.

Of course, in addition to the two basic models described above, there are sensing model that takes into account many other factors *(e.g. shadow-fading sensing model, Elfes sensing model)* [73].

In the *SAT*, *SMT* and graph-based models described later in this dissertation, we use the binary sensor model. It means that the sensing range for each node is a circular shape.

Environments and deployment strategies An important consideration is the topological deployment of nodes. The deployment method is application dependent and affects the performance of the routing protocol. The placement of the sensors can be either deterministic or random, but the final location depends largely on the attributes of the area or building, or simply on accessibility. While deterministic deployment takes into consideration the environmental parameters and the field of application to place the sensors strategically, the random method of deployment *(e.g. spreading from an air plane)* makes the sensors final position unpredictable. Another important trait of such networks is about the type of nodes it contains. We can differentiate homogeneous and heterogeneous nodes that made up a network [4, 5, 19]. In Figure 2.1 and 2.2 show a deterministic placement with heterogeneous sensor nodes *(different ranges)* and a random placement with homogeneous sensor nodes *(identical ranges)*. Depending on the environment the different types of *WSNs* include: Terrestrial, Underground, Underwater, Multimedia, Mobile. Overall, the most important goals of deployment strategies, depending on the environment and the area of application, to maximize the coverage and the connectivity and optimize the lifetime and the energy efficiency [1].

Lifetime A critical aspect of *WSN* applications is network lifetime [38]. It is perhaps the most important quality for the evaluation of *WSNs*. In Power-constrained *WSNs* the energy storage capacity of the nodes is finite. These networks are usable as long as they can communicate sensed data to a processing node. In 2002, Bhardwaj and Chandrakasan

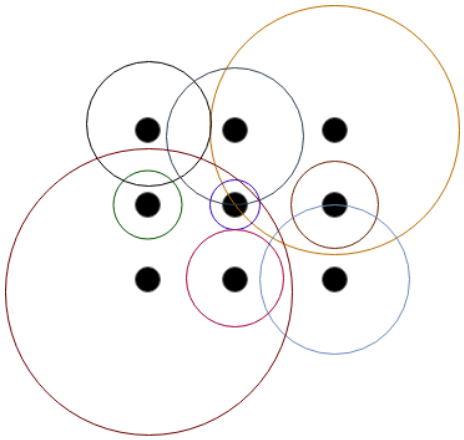


Figure 2.1: Deterministic placement with heterogeneous sensor nodes

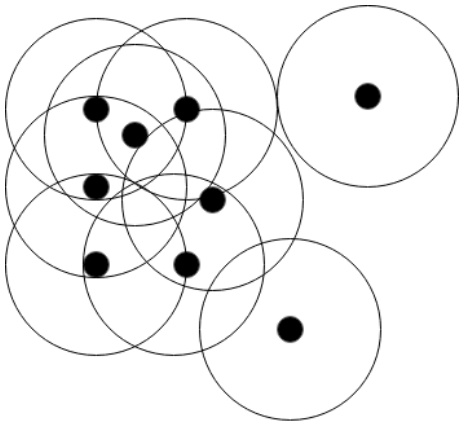


Figure 2.2: Random placement with homogeneous sensor nodes

proposed [20] the concept of *lifetime bound*. The lifetime t, achieved by a *WSN* is upper bounded thus:

$$t \leq \max_{c \in C} \quad t(c) \tag{2.2.4}$$

where denote feasible collaborative strategies by c and the *(infinite)* set of all such strategies by C. The lifetime achieved by c is denoted by $t(c)$. In 2004, Zhang et al. introduced [194] a new definition of network's lifetime, namely α-lifetime. The α-lifetime of the network has been defined by them as the interval during which at least α of the detection area is covered by at least one sensor node. In their model, only studied the relation between network's lifetime and the coverage of the sensing area. In 2005, Mo et al. [132] extend it the previous α-lifetime definition with probabilities. They defined the α-lifetime of a *WSN* as the expectation of the entire interval during which the probability of guaranteeing k-coverage of area A and the connectivity of the network simultaneously is at least α, where $0 < \alpha < 1$.

Coverage and connectivity Coverage and connectivity are the essential qualities of *WSNs*. Coverage of *WSN* means how well an area of observed is being monitored by the deployed sensors. The connectivity is also an easy-to-define concept. We say that there is a connection between two sensors if they can communicate with each other. These amounts are significantly depend on the previously described sensing modells and from different node types *(fixed or mobile nodes)*. We say that an observed T *(e.g. target)* point to be k-covered that if it is within at least k sensors sensing ranges. Maximizing the coverage and maintaining a lower cost of deployment have always been a challenge, especially when the monitoring area is unknown and hard to approachable. The purpose of deployment strategies is to ensure proper k-coverage while minimizing the use of energy consumption and interference. There are basically two deployment strategies: static coverage [159] and dynamic coverage [43, 120]. In this dissertation, we modelling static coverage problems

focusing on target discovery problem. In 2005 Cardei et al. [39] defined this special coverage problem class in which sense given m targets with known location and a randomly deployed energy-constrained *WSN* with n sensors the goal is how to schedule the sensor nodes activity such that all the targets are continuously observed and network lifetime is maximized. The target coverage problem is *NP*-complete [39].

Hop Count In a single-hop network there is only one *(single)* hop between the source nodes and the sink node, so they can directly communicate with each other. In a multi-hop network a sensor can also transmit data from the source to the sink because there are more than one hops from the source to the sink. In multi-hop networks one hop is the unit of the path between source and destination. The hop count refers to the number of intermediate nodes through which data must pass between source and destination. Networks can be classified by the number of hops between source nodes, which measures their environment, and a sink node, which collects data.

Architectures and Topologies The most important parameters of networks are reliability and scalability [5, 33]. Topological metrics commonly used on networks: number of nodes, number of edges, average degree, degree distribution, connectedness, diameter, number of independent paths. The most important parameters for measuring the effectiveness of *WSN*s are: scope and coverage, scalability, expected transmission number, number of hops, power consumption / lifetime. Since typically the *WSN*s are self-organized ad-hoc networks, most of the network structure is "unplanned". However, the design of logical topology is important, in addition to traditional topologies *(Star, Bus, Mesh)*, depending on the size of network and application protocol *(data gathering/collection, target tracking, routing, data aggregation, data dissemination)*, different underlying logical topologies have been used. The identified topologies are: flat topology, cluster-based topology, chain-based topology and tree-based topology [125, 165]. Flat and hierarchical topologies are the two most typical topologies of *WSN*s [96]. For flat topology there is no established logical structure, each node plays equal role in network formation, and must be involved in network control. This is a structure-free topology, the basis of many protocols such as data aggregation [188] and gathering [91], node scheduling [161] and routing [82] protocols. The hierarchical approach is a very popular and current architecture type in *WSNs*, which provides *WSNs* with scalability, ad-hoc, and fault tolerance and enables easy integration of applications into *WSNs*. In Figure 2.3 shows a basic hierarchical *WSN*. As you can see this basic approach consist of three layers: sensing layer, networking layer, and application layer [112].

Cluster-based topologies *(see Figure 2.5)* are the type of hierarchical topologies where a node *(dynamically or statically)* becomes a so-called cluster head (*CH)* to transmit the data collected from its environment to the base station *(e.g. through a backbone network of cluster heads)*. These topologies have widely been used for various types of protocols,

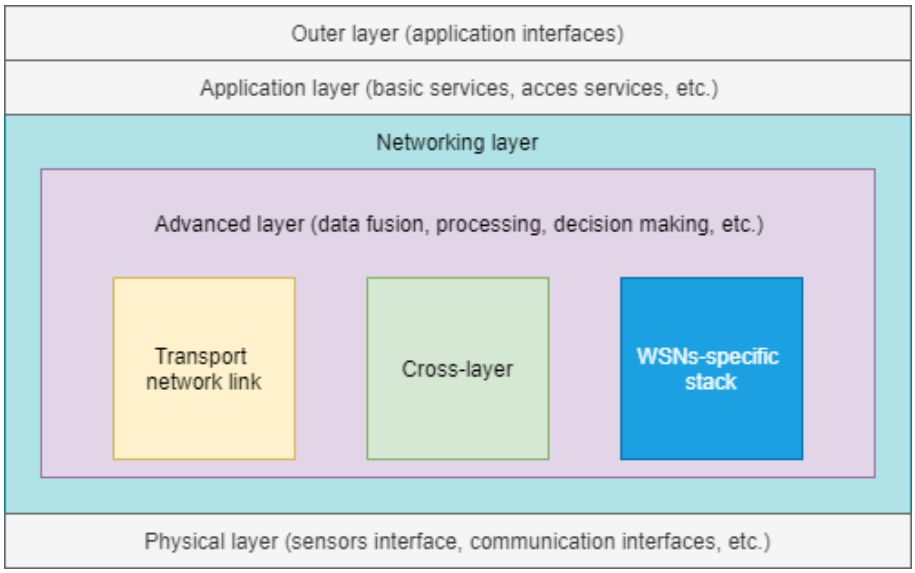


Figure 2.3: Hierarchical *WSNs*

such as data gathering [190], target tracking [42] and for many communication and routing protocols [5]. The advantages of cluster-based topology can be summarized in the following [177]:

- Clustering increases the life of the network by allowing the sensors to operate more energy efficiently with the optimal scheduling within the cluster.
- Dedication of *CHs* reduces communication redundancy and data aggregation inside the cluster.
- Within the cluster the size of routing table decreases, this reduces the resource demand for routing protocols.
- Due to intercluster-based communication, clustering conserves communication bandwidth.

Naturally, cluster-based topologies not only have advantages, but also have the disadvantages of the following:

- Additional overheads the creation of clusters and the selection of optimal *CHs*.
- The clustering size and cluster members of the clusters can vary from phase to phase, and their recalculation also results in extra computation overhead.

Routing Protocols Opposed to traditional networks, routing is one of the critical technologies in *WSNs*. Routing techniques are required for communication. Their basic task is to send data from sensor nodes to the base station. Two prominent classifications are known based on network structure and characteristic. Based on network structure the

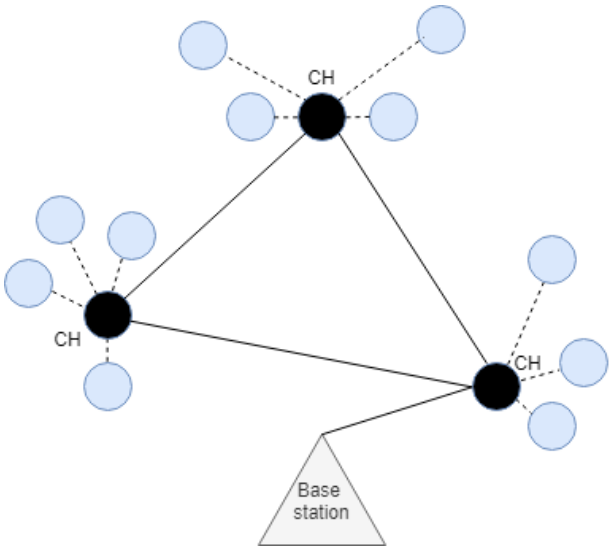


Figure 2.4: Two-tier clustering topology

most routing protocols can be classified as data-centric, hierarchical, location *(geographical)* or Quality of Service QoS [5, 4]. And, based on their basic characteristics can be classified as energy efficient, delay-less, secure, reliable protocols [162]. Over the past two decades a number of algorithms have been published to solve several routing issues. Most of these problems are related to optimization, such as energy saving and network lifetime problem [149, 173], cost of communication [123], collision probability minimization [127] and numberless routing optimization problem.

Protocol Overview In this subsection we give a brief summary of well-known clustering protocols in the field of *WSN*. There are many ways to classify the routing protocols of *WSN*. In Figure 2.5 shows an overview of the different classification methods of routing protocols.

From the point of view of sharing information *(processing of route)* in network distinguishes first *(direct)* and second-hand *(indirect)* information. The nodes disseminate this second hand information proactively *(table-driven)*, reactively *(on-demand)* or hybrid. In the case of proactive sharing of secondary information, the flow of information is continuous, the nodes sense the network on a regular basis and communicate the data sensed to the other nodes. The routing table needs to be updated continuously. In the case of reactive sharing, the constant update of routing table is not needed. Each node only searching/checking after a fixed time interval, or on the occurrence of some event or drastic change in the network [7, 93].

Depending on the routing structure of network routing protocols can be classified into the following three categories: flat, hierarchical and location-based. In flat topology, all nodes in the sensor network have equal roles in gathering information. They all have the same information about the state of the network. It is not possible to assign global identifiers to each node due to the large number of sensor nodes. In this type of rout-

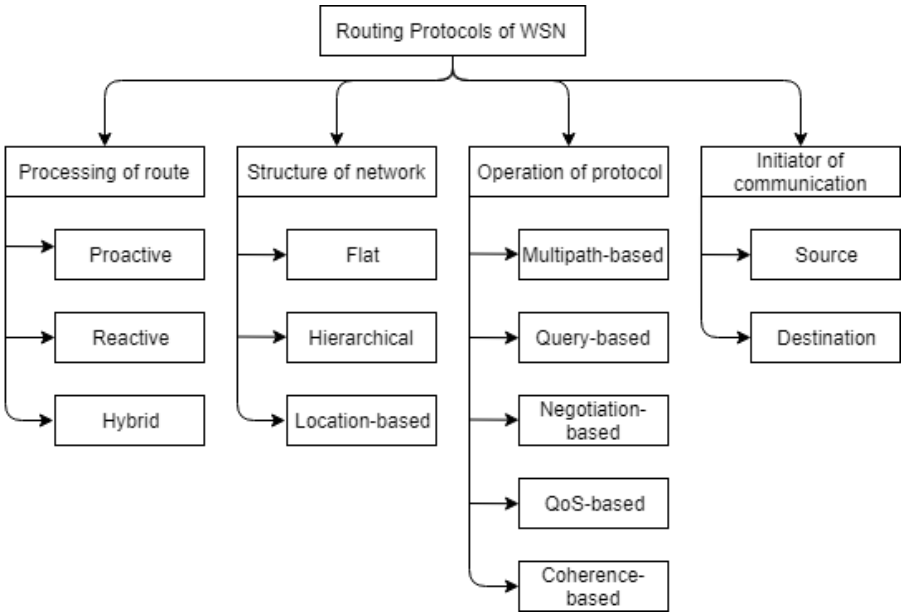


Figure 2.5: Classification of routing protocols

ing protocols have three types of flat routing schemes, namely, flooding, forwarding and data-centric based routing [55]. The most significant flat-based routing protocols are *COUGAR [189]*, *SPIN*-Sensor Protocols for Information via Negotiation [99], *DD*-Directed Diffusion [83], *RR*-Rumor Routing [32]. While in the case of the flat-based schemes each node has the same role, until then, in the hierarchical schema the nodes may have different roles. The hierarchical routing protocols must be designed in this light. Different systems are used different algorithms for the choice of *CHs* and creation of clusters. The most significant hierarchical-based routing protocols are *LEACH-Low-Energy Adaptive Clustering Hierarchy* [153], *PEGASIS*-Power-Efficient Gathering in Sensor Information System [119], *HEED*-Hybrid Energy-Efficient Distributed clustering [192], *TEEN*-Threshold Sensitive Energy Efficient Sensor Network Protocol [126]. In many cases, we need localization information about each node, for example their distance to adjust the power level of *RF* transceiver. Generally, two techniques are used to location, one is to localize the coordinate of the neighboring node and other is to use *GPS*. The most significant location-based-based routing protocols are *MECN*-Minimum Energy Communication Network [157], *GAF*-Geographic Adaptive Fidelity [69], *GEAR*-Geographic and Energy-Aware Routing [57]. Depending on the protocol operation, the routing protocols are divided into negotiation-based, query-based, multipath-based, coherent-based, and *QoS*-based routing. Multipath-based protocols are used multiple paths instead of single path to enhance network performance. In the case of query-based protocols, the station transmit a query for through the network example to find events/changes. Negotiation-based protocols are used to reduce redundant data transmissions through negotiation. In the target of *QoS*-based routing protocols to be balanced between energy consumption and data quality. And finally, in the case of coherent routing data is only transmitted to aggregators after

processing *(e.g. time stamping, duplicate suppression)* [140].

Topology Control One of the important property of an ad-hoc wireless network is node density. The dense layout makes the following properties available: high fault tolerance, high-coverage characteristics, but also causes some problems. The interference is high near to dense node areas, and there are a lot of collisions in case of message passing, which requires complicated operations for routing protocols, because of too many possible routes, routing needs lots of resources [28].

The aim of topology control techniques is reducing the cost of the distributed algorithms implemented on the network. The graph, which represents a network, has to be thinned because of cost-reduction by techniques like disconnection of nodes, removing links, changing scopes, etc., but the network-quality characteristics *(like scalability, coverage, fault tolerance, etc.)* must not fall below a required level [12, 111, 115, 148, 154].

Topology control algorithms are basically divided into two groups:

- *Tx*-based *(Gabriel Graph, Relative Neighbourhood Graph, Volonoi Diagram, LMST, iMST, Yao graph, distributed RNG, Kneight, etc.)*
- Hierarchical *(A3, EECDS, CDS-Rule K , HEED, LEACH, MESTER, SHORT, PEGASIS, ECHERP, etc.)*

Often the deployment is unattended, and randomly placed nodes are redundant. The aforementioned algorithms depending on the application area can be divided into two groups. The task of the first group of algorithms is to reduce redundancy and to create the appropriate topology. The second group of algorithms can be used in the so-called maintenance phase. Their aim is to optimally manage the reduced topology produced in the previous phase.

For flat networks with homogeneous nodes, the maximum energy savings can be achieved by changing *RF* transmitter power levels and adjusting nodes to sleep mode. Only by using these two procedures the number of connections *(interference)* can be significantly reduced.

Of course, after the reduction, it happens often that the fault-tolerant ability of the network falls below a critical level, this can be done by placing another node*(s)* in critical locations. For nodes with higher functionality *RF* transmitters, the number of connections can be further reduced by adjusting the characteristic. The result is a "thinner" topology and lower interference levels.

Controlling the performance of *RF* transmitter is also an important factor in hierarchical networks. Multilayer hierarchies are characterized by dynamically or statically assigned nodes such as cluster head, gateway, data collector, etc. Since here the network is divided into several clusters, it is sufficient locally to solve the optimal routing and performance control. The two most commonly used methods are: determination of dominant sets and clustering.

The overall aim is to create a scalable, fault-tolerant sparse topology, where the degree of the nodes is low, the maximum load is low, energy consumption is low and the paths are short. The following techniques are used to create an optimal topology: reducing the scope of nodes, removing some nodes, introducing a dominating set of nodes, clustering, and add some new nodes to gain all-all communication [160, 181, 111, 196].

2.3 Logic-Based Representations

Satisfiability

Propositional satisfiability is the problem of determining, for a formula of the propositional logic, if there is an assignment of truth value to its variables for which that formula evaluates to true. By *SAT* we mean the problem of propositional satisfiability for formulas in conjunctive normal form *CNF*. *SAT* is one of the most-researched *NP*-complete problems [49] in several fields of computer science, including theoretical computer science, artificial intelligence, hardware design, and formal verification [22]. Some particular cases of *SAT* are polynomially solvable: 2-*SAT*, Horn-*SAT*, Hidden Horn-*SAT*, *SLUR*.

The formula is 2-*SAT* (also called Binary *SAT* or Quadratic *SAT*), where each clause has exactly 2 literals. A canonical result about satisfiability theory is that the 2-*SAT* problem can be solved in linear time with graph representation-based methods, see Aspvall et al. [8].

A *CNF*-formula of which each clause contains at most k different literals is said to be a k-*CNF* formula. Example the below formula is a 2-*CNF* formula.

$$\phi = (\neg x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (\neg x_3 \vee \neg x_4) \tag{2.3.1}$$

A boolean formula is 3-*CNF* if each clause has at most three distinct literals. For example, the boolean formula

$$\phi = (\neg x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \tag{2.3.2}$$

is in 3-*CNF-SAT*. Any $(k \geq 3)$-*SAT* problem is *NP*-complete.

***DIMACS CNF* and *iCNF* Formats** Given a Boolean variable x, there exist two *literals*, the positive literal x and the negative literal $\overline{x}$. A *clause* is a disjunction of literals, a *cube* is a conjunction of literals. Either a clause or a cube can be considered as a finite set of literals. A *truth assignment* for a (finite) clause set or cube set F is a function ϕ that maps literals in F to {0,1}, such that if $\phi(x) = v$, then $\phi(\overline{x}) = 1 - v$. A clause respectively cube C is satisfied by ϕ if $\phi(l) = 1$ for some resp. every $l \in C$. A clause set respectively cube set F is satisfied by ϕ if ϕ satisfies C for every respectively some $C \in F$. For representing the input clause set for a *SAT* solver, the *DIMACS CNF* [1] format is commonly used, which

[1] `http://www.domagoj-babic.com/uploads/ResearchProjects/Spear/dimacs-cnf.pdf`

references a Boolean variable by its (1-based) index. At the top of the *DIMACS CNF* file is a simple header (*p* line).

```
p cnf <variables> <clauses>.
```

A negative literal is referenced by the negated reference to its variable. A clause is represented by a sequence of the references to its literals, terminated by a ”0”.
A simple *DIMACS CNF* file is the following:

```
p cnf 4 3
-1 2 3 0
-2 3 4 0
1 -3 4 0
2 3 4 -5 0
...
```

The *iCNF* [2] format extends the *CNF* format with a cube set. In this case the `p` line is simply `p inccnf`. A cube, called an assumption, is represented by a leading character ”a” followed by the references to its literals and a terminating ”0”.
A simple *iCNF* file is the following:

```
p inccnf
-2 -5 3 0
-3 1 0
a -1 0
...
```

State-of-the-art SAT Solvers Modern sequential *SAT* solvers are based on the Davis-Putnam-Logemann-Loveland (*DPLL*) [50] algorithm. This algorithm performs Boolean Constraint Propagation (*BCP*) and backtrack search, i.e., at each node of the search tree it selects a decision variable and assigns a truth value to it, then steps back when a conflict occurs. Conflict-driven clause learning *(CDCL)* [22] *(in Chapter 4)* is based on the idea that conflicts can be exploited to reduce the search space. If the method finds a conflict, then it analyzes this situation, determines a sufficient condition for this conflict to occur, in form of a learned clause, which is then added to the formula, and thus avoids that the same conflict occurs again. This form of clause learning was first introduced in the *SAT* solver `GRASP` [128] in 1996. Besides clause learning, lazy data structures are one of the key techniques for the success of *CDCL SAT* solvers, such as “watched literals” as pioneered in 2001, by the *CDCL* solver `Chaff` [134, 124] . Another important technique is the use of the `VSIDS` heuristics and the first-UIP backtracking scheme. In the state-of-the-art *CDCL* solvers, like `Lingeling` [23, 24], `Glucose` [13], `COMiniSatPS` [47] and `MapleCOMSPS` [116], several other improvements are applied. Besides enhanced preprocessing techniques, like e.g. failed literal detection, variable elimination, and blocked clause elimination, clause

[2] `http://users.ics.tkk.fi/swiering/icnf/`

deletion strategies and restart policies have a great impact on the performance of the *CDCL* solver. In 2016 Liang et al. [117] provided a new and even more effective branching heuristic than *VSIDS*, called the conflict history-based (*CHB*) branching heuristic. In the same year, a more efficient heuristics from the *CHB* and *VSIDS*, called learning rate branching (*LRB*) [116], were more frequently referred to. In recent years, machine learning based branching and restarts policies [118] have appeared.

Lookahead-based SAT Solvers Lookahead-based SAT solvers [22] *(in Chapter 5)* combine the *DPLL* algorithm with lookaheads, which are used in each search node to select a decision variable and at the same time to simplify the formula. One popular way of lookahead measures the effect of assigning a certain variable to a certain truth value: *BCP* is applied, and then the difference between the original clause set and the reduced clause set is measured *(by using heuristics)*. In general, the variable for which the lookahead on both truth values results in a large reduction of the clause set is chosen as the decision variable. The first lookahead *SAT* solver was `posit` [65] in 1995. It already applied important heuristics for pre-selecting the "important" variables, for selecting a decision variable, and for selecting a truth value for it. The lookahead solvers `satz` [107] and `OKsolver` [102] further optimized and simplified the heuristics, example `satz` does not use heuristics for selecting a truth value *(rather prefers* true*)*, and `OKsolver` does not apply any pre-selection heuristics. Furthermore, `OKsolver` added improvements like local learning and autarky reasoning. In 2002, the solver `march` [71] further improved the data structures and introduced preprocessing techniques. As a variant of `march`, `march_cc` [72] can be considered as a case splitting tool. It produces a set of cubes, where each cube represents a branch cutoff in the *DPLL* tree constructed by the lookahead solver. It is also worth to mention that `march_cc` outputs learnt clauses as well, which represent refuted branches in the *DPLL* tree. The resulting set of cubes represents the remaining part of the search tree, which was not refuted by the lookahead solver itself.

Parallel SAT Solvers There are two types of basic appearance of parallelism in computations, the "and-parallelism" and the "or-parallelism" [138]. The first is used in high performance computing, while the latter is more similar to nondeterministic guesses (data parallel). *SAT* can *(theoretically effectively)* be solved by several new computing paradigms using or-parallelism and by using, roughly speaking, an exponential number of threads. Since multi-core architectures are common today, the need for parallel *SAT* solvers using multiple cores has increased considerably.

In essence, there are two approaches to parallel *SAT* solving [68]. The first group of solvers typically follow a divide-and-conquer approach. They split the search space into several subproblems, sequential *DPLL* workers solve the subproblems, and then these solutions are combined in order to create a solution to the original problem. This first group uses relatively intensive communication between the nodes. They do for example

load balancing and dynamic sharing of learned clauses.

The second group apply portfolio-based *SAT* solving. The idea is to run independent sequential *SAT* solvers with different restart policies, branching heuristics, learning heuristics, etc. `ManySAT` [67] was the first portfolio-based parallel *SAT* solver. `ManySAT` applies several strategies to the sequential *SAT* solver `MiniSAT`. `Plingeling` [23, 24] follows a similar approach, and uses the sequential SAT solver `Lingeling`. In most of the state-of-the-art portfolio-based parallel *SAT* solvers (e.g. `ppfolio`, `pfolioUZK`, `SATzilla`) not only different strategies, but even different sequential solvers compete and, to a limited extent, cooperate on the same formula. In such approaches there is no load balancing and the communication is limited to the sharing of learned clauses.

`GridSAT` [45, 44] was the first complete and parallel *SAT* solver employing a grid. It belongs to the divide-and-conquer group. It is based on the sequential *SAT* solver `zChaff`. Besides achieving significant speedup in the case of some (satisfiable and even unsatisfiable) instances, `GridSAT` is able to solve some problems for which sequential `zChaff` exceed time out. `GridSAT` distributes only the short learned clauses over the nodes, therefore it minimizes the communication overhead. Search space splitting is based on the selection of a so-called pivot variable x on the second decision level, and then creating two subproblems by adding a new decision on x resp. $\neg x$ to the first decision level. If sufficient resources are available, the subproblems can further be partitioned recursively. Each new subproblem is defined by a clause set, including learned clauses, and a decision stack.

Hyvärinen et al. proposes a more sophisticated approach, based on using "partition functions", in order to split a problem into a fixed number of subproblems [74]. Two partition functions were compared, a scattering-based and a *DPLL*-based one with lookahead. A partition function can be applied even in a recursive way, by repartitioning difficult subproblems *(e.g. the ones that exceeds time out)*. For some of the experiments, an open source grid infrastructure called `Nordugrid` was used.

`SAT@home` [147] is a large volunteer SAT-solving project on grid, which involves more than 2000 clients. The project is based on the Berkeley Open Infrastructure for Network Computing `BOINC` [10], which is an open source middleware system for volunteer grid computing. On top of `BOINC`, the project was implemented by using the `SZTAKI Desktop Grid` [90], which provides the Distributed Computing Application Programming Interface (`DC-API`), in order to simplify the development, and then also to deploy and distribute applications to multiple grid environments. [147] proposes a rather simple partitioning approach: given a set of n selected variables, called a decomposition, a set of 2^n subproblems is generated. The key issue is how to select a decomposition. One way to solve this issue is to derive the set of "important" decomposition variables from the original problem formulation, which, however, then is problem-specific, and needs human guidance. For instance, in the context of *SAT*-based cryptoanalysis of keystream generators, a decomposition set can be obtained from the encoding of the initial state of the linear feedback shift registers [147]. `SAT@home` uses no data exchange among clients.

Satisfiability and Optimization Modulo Theories

Satisfiability Modulo Theories (*SMT*) is the decision problem of checking satisfiability of *(quantifier-free)* logical formulas with respect to some background theory. The *SMT*-based techniques exploit the advances in *SAT* solving for richer logics. The most common examples for theories are the integer numbers, the real numbers, the fixed-size bit-vectors, and the lists and the arrays.

SMT Solvers Whereas the language of `SAT` solvers is propositional calculus, the language of *SMT* solvers is predicate logic [56]. An *SMT* solver can solve a `SAT` problem, but not vice-versa. In the last 15 years, *SMT* solvers have attracted focused attention due to industrial applications. *SMT* solvers can be used in various fields of industrial applications *(e.g. analog circuit verification, verification of pipelined microprocessors, software verification and model checking, whitebox testing for security applications, etc.)*.

The history of the *SMT* solvers can be traced back to the seventies. There were two approaches for constructing decision procedures for combinations of ground theories [141, 167]. The modern *SMT* solvers started in the late 1990s with influenced by *SAT* solvers [128, 134]. Nowadays the most current *SMT* solvers (`Yices` [54], `Boolector` [34], `CVC3` [16], `MathSAT` [35], `Z3` [135]) use a *DPLL* based *SAT* solver.

SMT-LIB The `SMT-LIB` format [3] is a standard interface for *SMT* solvers. It provides standard descriptions of background theories and a common input and output languages for *SMT* solvers. Actually, the *SMT*-`LIB 2.6` logics refer to seven theories: (1) `ArraysEx`-Functional arrays with extensionality, (2) `FixedSizeBitVectors`-Bit vectors with arbitrary size, (3) `Core`-Core theory, defining the basic Boolean operators, (4) `FloatingPoint`-Floating point numbers, (5) `Ints`-Integer numbers, (6) `Reals`-Real numbers, (7) `Reals_Ints`-Real and integer numbers.

The logics that one could use might differ from each other in the linearity or non-linearity of arithmetic, the presence or absence of quantifiers, or in the presence or absence of uninterpreted functions.

We are using the quantifier-free logic of linear integer arithmetic with uninterpreted functions (`qfuflia`). The most important features are as follows:

- No quantifiers $\forall$ and $\exists$ are allowed to be used.
- Every expression must be of type integer or Boolean.
- Only the arithmetic operations addition, subtraction, multiplication, division, and comparison are to use.
- For the sake of linear arithmetic, expressions with multiplication are allowed to be used only in the format $c * t$ where c is a constant.

[3] http://smtlib.cs.uiowa.edu/

- It is allowed to use uninterpreted function symbols, i.e., to specify only the signature for such a function symbol.

Most of the *SMT* solvers support `qfuflia`, a `qfuflia` formula ϕ is satisfiable if there is an assignment of appropriate values to its variables and uninterpreted functions under which ϕ evaluates to true. Many problems of interest, which can be encoded as *SMT* problems, may require also to find models that are optimal with respect to some objective function [164].

Optimization Modulo Theories (*OMT*) is an extension of *SMT* with objective functions to maximize or minimize. An *OMT* problem, therefore, contains not only a formula to satisfy, but also an expression max : *obj* or min : *obj*, where *obj* denotes the objective function. There exist only a few *SMT* solvers that provide *OMT* solving, to the best of our knowledge: optimathsat [170], zthree [26], and symba [114].

The syntax for the optimization expression is not part of the *SMT-LIB* format and, therefore, is specified differently for the different *OMT* solvers.

2.4 Graph-Based Metrics and Representations

A directed graph $\mathcal{D} = (\mathcal{V},\mathcal{E})$ consists of the set $\mathcal{V}$ of nodes and the set $\mathcal{E}$ of edges, which are ordered pairs of elements of $\mathcal{V}$. A strongly connected digraph (*SCD*) is a directed graph in which it is possible to reach any node u starting from any other node v by traversing edges in the direction(s) in which they point. In the field of directed graphs, the problem to check strongly connectedness is a linear time problem [166]. The connectivity is one of the most central properties of graphs and has many applications [3, 28, 31, 58, 66, 143] for example in several fields of network analysis [2, 163, 174] and in the topology control of Wireless Sensor Networks [111].

Strong Connectivity In the 1950s and 1960s, mathematicians and computer scientists began to study the complexity of algorithms. Thus, in the case of graph-based algorithms, time and space complexity analysis became an important factor. It was Roy, who, in 1959, first considered the famous problem of computing the transitive closure *(finding all reachable vertices from each vertex)* of a directed graph [158]. In the 60s, a variety of sequential algorithms [183, 150] to solve this problem were proposed. Warshall solved the problem of computing the transitive closure of a binary relation represented as an adjacency matrix of the digraph and considered as a boolean matrix. The time complexity of the Warshall algorithm was $O(|\mathcal{V}|^3)$, then Purdom gave an $O(|\mathcal{V}|^2)$ algorithm. Munro [136] optimized Purdom's work in 1971 by using a more efficient data structure for merging the vertices. Instead of using an adjacency matrix for representing the edges, Munro notes the use of adjacency lists. As we know, the best algorithm [168] that solve this problem has $O(|\mathcal{V}| + |\mathcal{E}|)$ time complexity. The first "depth-first search" based algorithm

for finding the strongly connected components of a directed graph is presented in Tarjan's paper [175]. In the same year, R. Karp [92] introduced the intractability of the traveling-salesperson problem. Sharir [166] presented an algorithm for constructing all strongly-connected components of a directed graph. Like Tarjan, he used a linear time depth-first spanning tree algorithm. But this algorithm differed from Tarjan's in that it produced these components in reverse post-order of their roots, and also ordered the nodes within each component in reverse post-order. In both Tarjan's and Kosaraju's algorithms, there are fundamental techniques of efficient algorithm design for graphs. In 1982 Dijkstra proposed [52] a different variation of Tarjan's algorithm in order to find the maximum strong components in a directed graph. Instead of keeping track of low degree vertices, he maintains a stack of possible root candidates. On finding a last edge, the algorithm pops vertices from the stack until the 'root' of the cycle is found. At backtracking, the current flag of reachable states is set to false so that these states do not interfere with a future search. This algorithm also runs in linear time. Nearly forty years later H. Gabow [75] found a third linear-time algorithm for this problem in 2000. Contrary to earlier theories he presented a one-pass algorithm that only maintain a representation of the depth-first search path. This gives a simplified view of depth-first search without sacrificing efficiency.

Density of Graph In graph theory, the density of a graph $(\mathcal{V};\mathcal{E})$ can be calculated as $\frac{|\mathcal{E}|}{|\mathcal{V}|(|\mathcal{V}|-1)}$ [48]. Since the number of edges for a complete directed graph is $|\mathcal{V}|\,(|\mathcal{V}|-1)$, the maximum density is 1. Clearly, the minimum density is 0 *(for empty graphs)*. There is no strict distinction between sparse and dense graphs, and there is no exact definition for defining graph density. Streinu and Theran defined the concept of sparse and light graphs. A graph as being (k,l)-sparse if every nonempty subgraph with n vertices has at most $k \cdot nl$ edges, and (k,l)-tight if it is (k,l)-sparse and has exactly $k \cdot nl$ edges [171]. Nešetřil and Mendez defined the concept of somewhere dense and nowhere dense graph properties [142]. A graph a somewhere dense graph if exists a threshold t such that every complete graph appears as a t-subdivision in a subgraph of a graph. If such a threshold does not exist, the graph is nowhere dense.

Distance-Based Metrics The eccentricity of a node u is defined as the longest hop count between the node u and any other node in the graph. Centralization [63] is a general method for calculating a graph-level centrality score based on some node-level centrality metric. Centrality based metrics are the following ones: degree centrality *(based on degree)*, closeness centrality *(based on average distances)*, betweenness centrality (*based on geodesics)*, eigenvector centrality *(recursive: similar to page rank methods)*, eccentricity centrality.

Connection-Based Metrics The most basic connection-based metrics are the degree of a node, which is the number of edges to other nodes, and the degree distribution. The degree

distribution $P(k)$ of a graph is then defined to be the fraction of nodes in the network with degree k. Thus if there are n nodes in total in a graph and n_k of them have degree k, we have $P(k) = n_k/n$. Clustering is a fundamental and important property of networks, just like degree and degree distribution. Clustering coefficient is the measurement that shows the opportunity of a graph to be divided into clusters. Clusters are disjoint subgraphs of the graph. A cluster usually should be a complete subgraph, so in this way it is similar to a clique, but a cluster may consist of one node, on the other hand a clique is a complete subgraph which contains always at least two nodes in case of a communication graph. The clustering coefficient can globally [122, 184] or locally [185] characterize a graph. The global clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together. This measure is based on triplets of nodes. A triplet consists of three connected nodes [4]. The global clustering coefficient of a network, also known as transitivity T, which is the ratio of the number of loops of length three and the number of paths of length two.

The local clustering coefficient of vertex (u) in graph is given by $C_u = \frac{2e}{k(k-1)}$. Thus, C_u measures the ratio of the number of edges between the neighbors of u to the total possible number of such edges. The average clustering coefficient is the average of local clustering coefficients.

Prominent Graph Representations The most prominent graph representations are: Implication graph, And-Inverter Graph, Reduced Ordered Binary Decision Diagram, Zero-Suppressed Binary Decision Diagram.

- Implication graph [8] (IG) is a skew-symmetric directed graph, where vertices are literals, edges represent implication.
- And-Inverter Graph [81] (AIG) is directed acyclic graph where vertices are logical conjunction with two input edges, a marked edge means logical negation.
- Reduced Ordered Binary Decision (BDD) Diagram [36], which is a rooted, directed, acyclic graph consisting of vertices, which are boolean variables and terminal vertices, called 0-terminal, which terminates paths where the formula evaluates to false; and 1-terminal, which terminates paths, where the formula evaluates to true.
- *ZDD (called also ZBDD in the literature)*, Zero-Suppressed Binary Decision Diagram [131], is a kind of binary decision diagram. In *ZBDD* those vertices whose 1-edge points directly to 0-terminal are eliminated.

[4] https://www.geeksforgeeks.org/clustering-coefficient-graph-theory/

3 Part I

In this chapter, we present two *SAT* instance generators which can generate so-called *Black-and-White SAT* problems. A *Black-and-White SAT problem* has only two solutions which are opposite of each other.

3.1 Introduction

First in this chapter, we introduce the $WnDGen$ algorithm [210], which generates weakly nondecisive clause sets, which means that all clauses in the clause set are weakly nondecisive [103]. The notion of nondecisive clause [76] is a generalization of the notion of blocked clause [100]. A blocked clause can be added or removed from a clause set without changing its satisfiability. The same is true for nondecisive clauses. The notion of weakly nondecisive clause is a weakening of the notion of nondecisive clause, which means that each weakly nondecisive clause is also nondecisive. The notion of weakly nondecisive clause was introduced in [103] because blacked clause sets are always satisfiable, but the same is not true for nondecisive clause sets, so the author wanted to define a stronger notion than the blocked clause set but weaker than the nondecisive clause set which is still always satisfiable. This intention was not successful but it turned out that weakly nondecisive clause sets have interesting combinatorical properties, see in the next section, where we introduce also the formal definition of it.

The algorithm $WnDGen$ has two inputs, a clause and a number, its output is a clause set. In this chapter, we show that $WnDGen(C, k)$ is always satisfiable, and if $n \geq 2k - 3$ where n is the number of variables in C, then it has only two solutions: C and negation of C. This means that if C is the white clause or the black one then $WnDGen(C, k)$ is a black-and-white SAT problem, provided that $n \geq 2k - 3$. We also show that there is a threshold: Let n be the length of C; let S be the union of $WnDGen(C, k)$ and the set of C and negation of C; if $n \geq 2k - 3$, then S is unsatisfiable, if $n < 2k - 3$, then S is satisfiable. We show that around this threshold there are SAT instances, which are difficult for state-of-the-art SAT solvers, i.e., they are good for testing SAT solvers.

In the next part of this chapter, we introduce a propositional logic formula to model a directed graph and use a SAT solver to analyse it. This model is similar to the well-known one of Aspvall et al. models [8], but they create a directed graph from a 2-SAT problem, we generate a 2-SAT problem from a directed graph. In the Aspvall et al. model if the 2-SAT problem is unsatisfiable, then the generated directed graph is strongly connected. In

our model, if the directed graph is strongly connected, then the generated 2-*SAT* problem is a *Black-and-White 2-SAT* problem [198], which has two solutions: where each variable is true *(the white assignment)*, and where each variable is false *(the black one)*. If we see a directed graph as a communication model of a network, then we can ask in our model whether a node can send a message to another one through the network. More specifically we can ask whether all nodes can send messages to all other ones, i.e., the graph is strongly connected or not.

3.2 Generating Hard Combinatorical *SAT* Instances

In this section, we show how to generate hard combinatorical *SAT* instances. *SAT* Solving is a well-known area that has been actively researched for decades. However, in the last decade, thanks to the increase in the computing performance of computers, significant advances have been made in this area. Problems *(such as hardware and software verifications)*, which previously seemed unsolvable, are now easy to handle. New algorithms and heuristics, as well as increasingly sophisticated implementation techniques, all contributed to success. The state of the art *SAT* solvers are competing *(SAT Competition's [1])* every year, and the benchmarks to be solved are generally divided into three groups.

- *Applications* - both *SAT* and *UNSAT* problem encodings from real-world applications, such as hardware and software verification, bio-informatics, planning, scheduling, etc.
- *Hard Combinatorials* - both *SAT* and *UNSAT* hard combinatorial problems *(e.g. combinatorical problems, graph coloring problem, covering array, Ramsey number, Erdős's discrepancy conjecture, etc.)* to challenge current *SAT* solving algorithms.
- *Random SATs* - randomly generated satisfiable instances.

3.2.1 Notations

In the rest of the thesis, let V be a finite *set of Boolean variables*. V is the set of variables.

- The *negation of a variable* v is denoted by $\overline{v}$. *Negation of negation* is defined as $\overline{\overline{v}} := v$.

  The *negation of a set* U is denoted by $\overline{U}$ and defined by $\overline{U} := \{\overline{u} \mid u \in U\}$. *Negation of negation of a set* is defined as $\overline{\overline{U}} := U$.
- *Variables of a set* C is denoted by $\mathbf{V}ar(C)$ and defined as $\mathbf{V}ar(C) := (C \cup \overline{C}) \cap V$.
- *Literals* are the members of the set $W := V \cup \overline{V}$.
- *Positive literals* are the members of the set V.

[1] `http://www.satcompetition.org`

- *Negative literals* are their negations. If w denotes a negative literal $\overline{v}$, then $\overline{w}$ denotes the positive literal v.
- *Clauses* and *assignments* are finite sets of literals that do not contain simultaneously any literal together with its negation. A *clause set* is a finite set of clauses. The B, C, D, and E are clauses; S is a clause set.
- The *length* of a set U is its cardinality, denoted by $|U|$. Let $n := |V|$.
- If $|C| = k$, then we say that C is a *k-clause*. Special cases are *unit clauses* or *units* which are 1-clauses, and *full-length* or *clear clauses* which are n-clauses.
- We define the notion that C is *subsumed by* S, denoted by $C \supseteq\in S$, as follows:

$$C \supseteq\in S :\Longleftrightarrow \exists[B \in S]B \subseteq C. \tag{3.2.1}$$

- An assignment M is a *model* or *solution* of S iff for all $C \in S$ we have $M \cap C \neq \emptyset$. S is *satisfiable* iff it has a model, otherwise it is *unsatisfiable*.
- The *clause difference* of C and D, denoted by $diff(C, D)$, is defined as $diff(C, D) := C \cap \overline{D}$. If $diff(C, D) \neq \emptyset$ then we say that C *differs from* D.
- *Resolution rule can be performed* on two clauses iff they differ only in one variable. If resolution rule can be performed then the *resolvent*, denoted by $Res(C, D)$, is defined as:

$$Res(C,D) := (C \cup D) \setminus (diff(C,D) \cup diff(D,C)). \tag{3.2.2}$$

- *Blocked literal* c in clause C and in clause set S is denoted by $Blck(c, C, S)$ and defined as follows. If $c \in C \wedge C \in S$, then:

$$Blck(c, C, S) :\Longleftrightarrow \forall[B \in S][\overline{c} \in B]\exists[b \in B][b \neq \overline{c}]\overline{b} \in C \tag{3.2.3}$$

- *Blocked clause* C in S is denoted by $Blck(C, S)$ and defined as follows. If $C \in S$, then:

$$Blck(C, S) :\Longleftrightarrow \exists[c \in C]Blck(c, C, S). \tag{3.2.4}$$

- *Blocked clause set* S is denoted by $Blck(S)$ and defined as follows:

$$Blck(S) :\Longleftrightarrow \forall[C \in S]Blck(C, S). \tag{3.2.5}$$

- *Nondecisive literal* c in clause C and in clause set S is denoted by $NonD(c, C, S)$ and defined as follows. If $c \in C \wedge C \in S$, then:

$$\begin{aligned} NonD(c, C, S) :\Longleftrightarrow \forall[B \in S][\overline{c} \in B](\exists[b \in B][b \neq \overline{c}]\overline{b} \in C \vee \\ Res(C, B) \cup \{c\} \supseteq\in S \setminus \{C\}) \end{aligned} \tag{3.2.6}$$

- *Nondecisive clause* C in S is denoted by $NonD(C,S)$ and defined as follows. If $C \in S$, then:

$$NonD(C,S) :\Longleftrightarrow \exists[c \in C]NonD(c,C,S). \tag{3.2.7}$$

- *Nondecisive clause set* S is denoted by $NonD(S)$ and defined as follows:

$$NonD(S) :\Longleftrightarrow \forall[C \in S]NonD(C,S). \tag{3.2.8}$$

- *Weakly nondecisive literal* c in clause C and in clause set S is denoted by $WnD(c,C,S)$ and defined as follows. If $c \in C \wedge C \in S$, then:

$$WnD(c,C,S) :\Longleftrightarrow \forall[B \in S][\overline{c} \in B](\exists[b \in B][b \neq \overline{c}]\overline{b} \in C \vee Res(C,B) \supseteq\in S) \tag{3.2.9}$$

- *Weakly nondecisive clause* C in S is denoted by $WnD(C,S)$ and defined as follows. If $C \in S$, then:

$$WnD(C,S) :\Longleftrightarrow \exists[c \in C]WnD(c,C,S). \tag{3.2.10}$$

- *Weakly nondecisive clause set* S is denoted by $WnD(S)$ and defined as follows:

$$WnD(S) :\Longleftrightarrow \forall[C \in S]WnD(C,S). \tag{3.2.11}$$

In other words, a literal is weakly nondecisive in C and S iff it is blocked or the resolvent of C and a non-blocking B is subsumed by S. C is a weakly nondecisive clause in S iff it has a weakly nondecisive literal in C and S. S is a weakly nondecisive clause set iff its clauses are weakly nondecisive.

3.2.2 Generating Weakly Nondecisive Clause Sets

$WnDGen(C,k)$ works as follows: It generates all k-length subsets of C. For each subset it adds k clauses to the output, negating every time another literal is in the subset. Then it does the same with the negation of C. We show that the resulting clause set is always weakly nondecisive and satisfiable. Actually, C and negation of C are solutions of the SAT instance generated by $WnDGen(C,k)$.

We also show that there is a threshold: Let n be the length of C; let S be the union of $WnDGen(C,k)$ and the set of C and negation of C; if $n \geq 2k-3$, then S is unsatisfiable, if $n < 2k-3$, then S is satisfiable.

3.2.3 Definitions of $WnDGen1$ and $WnDGen$

We introduce two generator functions: $WnDGen1$ and $WnDGen$, which can generate weakly nondecisive clause sets.

The first generator function is $WnDGen1$. If C is a clause, k is a natural number, and $2 \leq k \leq |C|$, then

$$WnDGen1(C,k) := \{D \mid \mathbf{V}ar(D) \subseteq \mathbf{V}ar(C) \ \wedge |D| = k \ \wedge \ |diff(D,C)| = 1\}. \tag{3.2.12}$$

This means that $WnDGen1(C, k)$ generates all k-clauses which differ only in one literal from C. C is called the generator clause.

We give two examples for $WnDGen1$:

$$WnDGen1(\{a, b, c\}, 2) = \{\{a,\overline{b}\}, \{\overline{a},b\},\{a,\overline{c}\},\{\overline{a},c\},\{b,\overline{c}\},\{\overline{b},c\}\}. \tag{3.2.13}$$

$$WnDGen1(\{a, b, c\}, 3) = \{\{a,b,\overline{c}\}, \{a,\overline{b},c\},\{\overline{a},b,c\}\}. \tag{3.2.14}$$

The second generator function is $WnDGen$. If C is a clause, k is a natural number, and $2 \leq k \leq |C|$, then:

$$WnDGen(C, k) := WnDGen1(C,k) \cup WnDGen1(\overline{C}, k). \tag{3.2.15}$$

3.2.4 Properties of $WnDGen1$ and $WnDGen$

In this section, we prove that $WnDGen1$ and $WnDGen$ generate weakly nondecisive clause sets. We give a criterion on the shape of clauses which are subsumed by $WnDGen1$. Based on this result we show how to use $WnDGen$ to generate unsatisfiable clause sets.

Shape criterion

First we give the shape criterion.

Lemma 1. *Assume C and D are clauses and $\boldsymbol{V}ar(D) \subseteq \boldsymbol{V}ar(C)$. Assume k is a natural number and $2 \leq k \leq |C|$. If $|diff(D, C)| \geq 1$ and $|D| \geq |diff(D, C)| + k - 1$, then D is subsumed by $WnDGen1(C, k)$.*

Proof. Without loss of generality we may assume that C contains only positive literals. Then $WnDGen1(C, k)$ contains all clauses with $k - 1$ positive literals and 1 negative one. Since $|diff(D, C)|$ is the number of negative literals in D and $|D| - |diff(D, C)|$ is the number of positive literals in D we have that D is subsumed by $WnDGen1(C, k)$ if $|diff(D, C)| \geq 1$ and $|D| - |diff(D,C)| \geq k - 1$. □

Note, that there is a special case. If $|diff(D, C)| = 1$ then it is enough to show that $|D| \geq k$ to prove that D is subsumed by $WnDGen1(C, k)$. We will use this case in Lemma 2. We use another case in Lemma 3 when $|diff(D, C)| = k - 1$. In this case it is enough to show that $|D| \geq 2k - 2$.

$WnDGen1$ is a weakly nondecisive clause set

Lemma 2. *Assume C is a clause, k is a natural number and $2 \leq k \leq |C|$. Let $S = WnDGen1(C, k)$. Then S is a weakly nondecisive clause set.*

Proof. Assume C is a clause, k is a natural number, $2 \leq k \leq |C|$ and $S = \mathrm{WnDGen1}(C, k)$. We show that S is a weakly nondecisive clause set. To show this, by the definition of

weakly nondecisie clause set, we assume that A is an arbitrary but fixed clause in S, and we show that A is a nondecisive clause in S. To show this, by the definition of nondecisieve clause, we have to find a literal a' which is a weakly nondecisive literal in A and S. Since A is an element of $WnDGen1(C,k)$, from its definition, we know that $|diff(A,C)| = 1$, i.e., there is a literal $\overline{a}$ in C such that a is in A and there are no other opposite literals in A and C. We show that a is a weakly nondecisive literal in A and S, i.e., it is a suitable choice for a'. To show this, by the definition of weakly nondecisive literal, we have to show that $Res(A,B)$ is subsumed in S for some arbitrary but fixed clause B from S which contains $\overline{a}$; or there is an another literal in B which occurs oppositely in A. Assume that there is no such literal, i.e., we have to show that $Res(A,B)$ is subsumed in S. To show this, by Lemma 1, we have to show that $|diff(Res(A,B), C)| = 1$ and $|Res(A,B)| \geq k$. From the definition of resolution, we know that neither a nor $\overline{a}$ is in $Res(A,B)$. We know that $\overline{a}$ is in B, since B is a clause, from the definition of clause and we also know that a is not in B. Since B is an element of $WnDGen1(C,k)$, from its definition, we know that there is a literal $\overline{b}$ in C such that b is in B and there are no other opposite literals in B and C. We already know that $\overline{a}$ in C such that a is in A and there are no other opposite literals in A and C. From these two assumptions, we obtain that $\overline{b}$ is in $Res(A,B)$, and there is no other opposite literal in $Res(A,B)$ and C, i.e., $|diff(Res(A,B), C)| = 1$. Now we have to show that $|Res(A,B)| \geq k$. From the definition of resolution, we know that $|Res(A,B)| \geq k-1$. We show that $|Res(A,B)| \neq k-1$. We also show this by contradiction. Assume $|Res(A,B)| = k - 1$, but in this case b must be an element of A, which means that $diff(A,C) = \{a, b\}$, which contradicts the assumption that $|diff(A,C)| = 1$. So we obtain that $|Res(A,B)| \geq k$. This means that $Res(A,B)$ is subsumed in S. This means that a is a weakly nondecisive literal in A and S. Hence, S is a weakly nondecisive clause set. □

This was a relatively long technical proof. A shorter alternative proof can be found in [210].

Actually, we can state a more powerful statement, that in $WnDGen1(C, k)$ all literals of all clauses are weakly nondecisive, see Lemma 4. But this weaker lemma helps us to prove that $WnDGen(C,k)$ is also weakly nondecisive.

$WnDGen$ is a weakly nondecisive clause set

Lemma 3. *Assume C is a clause, k is a natural number and $2 \leq k \leq |C|$. Let $S = WnDGen(C, k)$. Then S is a weakly nondecisive clause set.*

Proof. Let $S = WnDGen(C, k)$. We show that S is weakly nondecisive. To show this, we show that any A in S is a weakly nondecisive clause in S. Without loss of generality we may assume that A is an element of $WnDGen1(C, k)$, this means that $|A| = k$ and $|diff(A,C)| = 1$. The main idea of the proof is that there is a literal a, which occurs positively (negatively) in A, but negatively (positively) in C and this literal is a nondecisive

literal in A and S. To show this, we show that for any B in S such that which contains $\overline{a}$ we have that either B blocks A (i.e., $\exists[b \in B][b \neq \overline{a}]\overline{b} \in A$) or $Res(A,B)$ is subsumed in S. We know from Lemma 2 that this is true if B is an element of $WnDGen1(C,k)$. If B is in $WnDGen1(\overline{C},\ k)$ then we have $|diff(B,\overline{C})| = 1$, i.e., $|diff(B,C)| = k-1$ and we also know that $|diff(A,C)| = 1$ which means that there is only one case when B does not block A, when $|\mathbf{V}ar(A) \cap \mathbf{V}ar(B)| = 1$, but in this case, by Lemma 1 $Res(A,B)$ is subsumed in S, because $|Res(A,B)| = 2k-2$ and $|diff(Res(A,B),C)| = k-1$. Hence, S is weakly nondecisive. □

All literals in $WnDGen1(C,k)$ are weakly nondecisive

Now we prove that all literals in $WnDGen1(C,k)$ are weakly nondecisive.

Lemma 4. *Assume C is a clause, k is a natural number and $2 \leq k \leq |C|$. Let $S = WnDGen1(C,\ k)$. Then for all C in S we have that all literals in C and S are weakly nondecisive.*

Proof. Without loss of generality we may assume that C contains only positive literals. From definition of $WnDGen1$ we know that in each clause in S there is only one negative literal, this is the literal in which each clause differs from C. From the proof of Lemma 2 we know that these literals are weakly nondecisive. This means that all negative literals in S are weakly nondecisive. From this, we can obtain that all positive literals are also weakly nondecisive, because if a positive literal is involved in a resolution, there must be involved also a negative one, and the resolvent will be subsumed because the negative literal is weakly nondecisive. Hence, all literals in S are weakly nondecisive. □

$WnDGen(C,k)$ is always satisfiable

We proof now that $WnDGen(C,k)$ is always satisfiable, which means naturally that $WnDGen1(C,k)$ is always satisfiable.

Lemma 5. *Assume C is a clause, k is natural number and $2 \leq k \leq |C|$. Then $WnDGen(C,k)$ is satisfiable.*

Proof. It is easy to see that for all $D \in WnDGen(C,k)$ we have that $D \cap C \neq \emptyset$ and $D \cap \overline{C} \neq \emptyset$. This means that C and $\overline{C}$ are models for $WnDGen(C,k)$. Hence, $WnDGen(C,k)$ is satisfiable. □

How to generate unsatisfiable SAT instances by the help of $WnDGen(C,k)$

We have seen that $WnDGen(C,k)$ is satisfiable. Now let us see what we should add to it to make it unsatisfiable. The next lemma is based on the *Clear Clause View* which was introduced in [103]. This view describes a clause set as the set of subsumed full-length clause. One of its basic ideas is that a clause set is unsatisfiable if and only if it subsumes all full-length clauses, see Lemma 4.7.1 in [103].

Lemma 6. *Assume C is a clause, k is natural number and $2 \leq k \leq |C|$. Then $WnDGen(C,k) \cup \{C, \overline{C}\}$ is unsatisfiable if and only if $|C| \geq 2k-3$.*

Proof. Let $V = \mathbf{V}ar(C)$, i.e., $n = |C|$. Without loss of generality we may assume that C contains only positive literals.
($\Leftarrow$) We assume that $n \geq 2k-3$. We show that $WnDGen(C,k) \cup \{C, \overline{C}\}$ is unsatisfiable. To show this, by Lemma 4.7.1 in [103], it is enough to show that $WnDGen(C,k) \cup \{C, \overline{C}\}$ subsumes all full-length clauses. Any full-length clause has either $0, 1, \ldots, n-1$ or n negative literals. C subsumes those full-length clauses which have 0 negative literals (actually there is only one such clause: C itself). Furthermore, $\overline{C}$ subsumes those full-length clauses which have n negative literals (because it subsumes itself). Therefore, we have to show that $WnDGen(C,k)$ subsumes all full-length clauses with 1, 2, ..., $n-2$, or $n-1$ negative literals. Let D be a full-length clause such that $D \neq C$. This means that $|diff(D,C)| \geq 1$, and, by definition of full-length clause, $|D| = n$. From Lemma 1 we know that D is subsumed by $WnDGen1(C,k)$ if $|D| \geq |diff(D,C)| + k - 1$, i.e., $n - k + 1 \geq |diff(D,C)|$. Note, that the number $|diff(D,C)|$ is the number of negative literals in D since C contains only positive literals. This means that $WnDGen1(C,k)$ subsumes those full-length clauses which contain $1, 2, \ldots, n-k$, or $n-k+1$ negative literals. From this, by definition of $WnDGen$, we can obtain that we have to show that $WnDGen1(\overline{C},k)$ subsumes those full-length clauses which contain $n-k+2, n-k+3, \ldots, n-2$, or $n-1$ negative literals; on the other way around, we have to show that $WnDGen1(\overline{C},k)$ subsumes those full-length clauses which contain $n-(n-k+2), n-(n-k+3), \ldots, n-(n-2)$, or $n-(n-1)$ positive literals, i.e., $k-2, k-3, \ldots, 2$, or 1 ones. From Lemma 1 we know that $WnDGen1(\overline{C},k)$ subsumes those full-length clauses which contain $1, 2, \ldots, n-k$, or $n-k+1$ positive literals. This means that we have to show that the number of cases which are subsumed by WnDGen1($\overline{C}$, k) is greater than or equal to the number of cases which remains un-subsumed by WnDGen1(C, k), i.e., $n-k+1 \geq k-2$. We already know that $n \geq 2k-3$, from this we obtain that $n-k+1 \geq k-2$. This means that $WnDGen(C,k)$ subsumes all full-length clauses with 1, 2, ..., $n-2$, or n-1 negative literal. Hence, $WnDGen(C,k) \cup \{C, \overline{C}\}$ is unsatisfiable.
($\Rightarrow$) We assume $WnDGen(C,k) \cup \{C, \overline{C}\}$ is unsatisfiable. We show $n \geq 2k-3$. From the assumption, by Lemma 4.7.1 in [103], we obtain that $WnDGen(C,k) \cup \{C, \overline{C}\}$ subsumes all full-length clauses. This means that $WnDGen(C,k)$ subsumes all full-length clauses with 1, 2, ..., or $n-1$ negative literals. Note, that for an arbitrary clause D the number $|diff(D,C)|$ is the number of negative literals in D since C contains only positive literals. From Lemma 1, we can obtain that $WnDGen1(C,k)$ subsumes those full-length clauses which contain $1, 2, \ldots, n-k$, or $n-k+1$ negative literals. We can also obtain that $WnDGen1(\overline{C},k)$ subsume those full-length clauses which contain $n-1, n-2, \ldots, n-(n-k)$, or $n-(n-k+1)$ negative literals. This means that both $WnDGen1(C,k)$ and $WnDGen1(\overline{C},k)$ subsumes $n-k+1$ cases from the sequence $1, 2, \ldots,$

or $n-1$, but start from different ends. From our assumption we know that there is no such case which is not subsumed from this sequence. This means that the number of subsumed cases must be greater than or equal to the number of cases in the sequence, i.e., $2*(n-k+1) \geq n-1$. After simplification, we obtain $n \geq 2k-3$. Hence, $|C| \geq 2k-3$. □

The Lemma 6 was a kind of surprise for us, because we had a feeling that $WnDGen$ with the generator clause and with the negation of the generator clause should be unsatisfiable. It turned out that sometimes this structure is satisfiable. This lemma gives us a threshold, $2k-3$, such that below that this structure is satisfiable.

$WnDGen(C,k)$ can generate a Black-and-White clause set

In Section 3.3, we introduce formally the notion of *Black-and-White* clause set but here we have to note that $WnDGen(C,k)$ can generate a *Black-and-White* clause set. The white *(black)* clause / assignment is the full-length clause / assignment which contains only positive *(negative)* literals. A *Black-and-White* clause set has only two solutions, the black and the white assignments. This notions will be defined more formally in Section 3.3.

Lemma 7. *Assume C is the white or the black clause, k is natural number, and $2 \leq k \leq |C|$. Then $WnDGen(C,k)$ is a Black-and-White clause set if and only if $|C| \geq 2k-3$.*

Proof. This Lemma is an easy consequence of Lemma 5 Lemma 6 and the definition of *Black-and-White* clause. □

3.2.5 Test Results

We had the idea that Lemma 6 gives us a threshold and be the generated SAT instances around this threshold are may be interesting. It turned out that on the threshold, i.e., if $|C| = 2k-3$, the generated SAT instances are hard to solve for state-of-the-art SAT solvers. We developed a so called $WnDGen$ tool which can generate weakly nondecisive clause sets with the $WnDGen$ algorithm. This tool can be downloaded from `http://fmv.ektf.hu`. Our tests were done on Intel quadcore machine with 6 GB of RAM running at 2.66 GHz.

For testing, we used the `MiniSat`[2] 2.2.0. The `MiniSat` is a modern *CDCL* solver has four major features: - Conflict-driven clause learning [128], [129], - Random search restarts [78], - Boolean constraint propagation using lazy data structures [133], - Conflict-based adaptive branching [133]. For each instance, we used a timeout of 1 hour.

[2] http://www.minisat.se

WnDGen	restarts	conflicts	decisions	prop.	conflict lit.	CPU time(s)
n=5 k=4	1	1	4	7	3	0.00
n=7 k=5	1	9	12	30	30	0.00
n=9 k=6	1	31	34	90	124	0.00
n=11 k=7	2	105	111	304	524	0.01
n=13 k=8	3	283	297	796	1812	0.04
n=15 k=9	6	1502	1533	4320	11028	13.64
n=17 k=10	9	6335	6406	18510	52287	9.61
n=19 k=11	12	24960	25089	72661	229711	157.14
n=21 k=12	15	78298	78472	222964	812344	2779.43

Table 3.1: Runtimes and results (all instances are *sat*)

The result are presented in Table 3.1 and Table 3.2. The first column presents the $WnDGen$ instances. The second column shows the number of restarts. The next four columns give information about number of conflicts, decisions, propagations and conflict literals. Finally, the last column shows *CPU* times. The tables show *(e.g. CPU time)* that the problem generated by the $WnDGen$ algorithm explodes exponentially as a function of n and k.

WnDGen	restarts	conflicts	decisions	prop.	conflict lit.	CPU time(s)
n=5 k=4	1	8	7	18	12	0.00
n=7 k=5	1	22	21	54	49	0.00
n=9 k=6	1	72	71	192	228	0.00
n=11 k=7	3	256	257	705	1056	0.01
n=13 k=8	5	935	945	2651	4805	0.11
n=15 k=9	8	3481	3514	10148	21544	1.92
n=17 k=10	11	12960	13029	37004	92650	30.45
n=19 k=11	14	49168	49296	140618	399670	448.03
n=21 k=12	TIME OUT					

Table 3.2: Runtimes and results *(all instances are unsat)*

3.3 *SAT* Representation of Wireless Sensor Networks

In this section, we study how to represent a directed graph as a propositional formula. In our model, vertices are boolean variables and edges are implications. This means that our model is similar to an implication graph, but in the case of implication graphs vertices are literals. The intuition behind our model comes from the field of *WSN*, where it is a relevant problem whether each sensor can communicate with all other ones through the network. If the network is represented by a directed graph where vertices are the sensors, and an edge represents that a sensor can send data to an other one, then this problem boils down to checking whether the graph is strongly connected.

We wanted to solve this problem by a *SAT* solver, so we had to convert the above directed graph into a *SAT* problem. Since in our model each edge represents a logical implication we can generate a 2-*SAT* problem where each clause contains exactly one positive and one negative literal.

We have found that the graph is strongly connected if this 2-*SAT* problem has exactly two solutions: the first is the one where each boolean variable is true *(which is called the white assignment)*, the second is the one where each boolean variable is false *(which is called the black assignment)* [198]. Such a *SAT* problem is called a *Black-and-White SAT* problem.

This means that if we add the negation of these two solutions *(the black one and the white one)* to the generated *SAT* problem, then it will be unsatisfiable and will be not 2-*SAT* any more.

In other words, a *SAT* problem is a *Black-and-White SAT* problem if and only if it is satisfiable and has only two solutions, the white assignment and the black one. So it becomes unsatisfiable if we add the negation of these assignments, which are two full-length clauses, the clause which contains only negative literals *(which is called the black clause)*, and the clause which contains only positive literals *(which is called the white clause)*.

In the field of directed graphs, the problem to check strongly connectedness is a linear time problem [166]. The Black-and-White 2-*SAT* problem is also a linear time problem as we are going to show that later in this section. The question arises, while should we transform a graph into a *SAT* problem to check a property which can be checked in linear time in both fields? We think that our model is a new link between the two fields: We are going to prove that *Black-and-White 2-SAT* problems and strongly connected graphs are equivalent. The *Black-and-White SAT* problem appears also as a special case of weakly nondecisive *SAT* problems, see Lemma 6 in [210] *(in that paper we did not use the term Black-and-White SAT problem, this term is introduced in this section)*. This suggests two things: the *Black-and-White SAT* problem could be an interesting problem in general, and since there are weakly nondecisive 3-*SAT* problems, which are also *Black-and-White*, there might be a 3-*SAT* representation of directed graphs.

3.3.1 The logical Model of a Directed Graph

Let $\mathcal{D} = (\mathcal{V},\mathcal{E})$ be a directed graph, where $\mathcal{V}$ is the set of vertices, and E is the set of edges. We say that $\mathcal{D}$ is a communication graph if and only if the elements of $\mathcal{V}$ are atomic formulas or each vertex is labeled by a different atomic formula. Note, that any directed graph is a communication graph because we can label each vertex by a different atomic formula. In this section, we assume that the elements of $\mathcal{V}$ are atomic formulas. In other words, if x is an element of $\mathcal{V}$, then for example $\neg x$ must not be an element of $\mathcal{V}$. From a communication graph we create the following model: We represent vertices by boolean variables and edges by logical implication. The conjunction of these formulas gives the logical model of the directed graph. For example, if the vertex x_1 has edges to vertices x_2 and x_3, then the logical model is:

$$(x_1 \supset x_2) \wedge (x_1 \supset x_3). \tag{3.3.1}$$

This formula can be easily transformed to a 2-SAT problem by rewriting implications by the rule $x \supset y = \neg x \vee y$. Note, that although each edge in $\mathcal{D}$ is interpreted as a logical implication, it is not an implication graph, because it does not contain negative literals. In other words, our model is not the same used by Aspvall et al. in [8]. They create a graph from a 2-SAT problem, we generate a 2-SAT problem from a communication graph. In their directed graph, each variable is represented by a positive and a negative literal. In our case, we have only positive literals in the directed graph.

We give the definition of our model in a more formal way. The 2-SAT representation $\mathcal{M}$ of a communication graph $\mathcal{D}$ is defined as follows:

$$\mathcal{M} := \bigwedge_{i=1,j=1,i\neq j}^{n} \mathcal{P}(x_i,x_j) \tag{3.3.2}$$

$$\mathcal{P}(x,y) = \begin{cases} \neg x \vee y, & if\ x\ has\ an\ edge\ to\ y \\ True & otherwise \end{cases} \tag{3.3.3}$$

Note, that $\mathcal{M}$ is a 2-SAT problem and has a nice property, each clause in it has exactly one positive and one negative literal therefore $\mathcal{M}$ is satisfiable. It has at least two solutions: one where each variable is true which is called the white assignment, and one where each variable is false which is called the black assignment.

If $\mathcal{M}$ has only these two solutions, and no other one, then $\mathcal{D}$ has an interesting property, namely it is strongly connected. To check this, we need the white-or-black constraint, which is the disjunction of the white assignment and the black assignment, that states that all vertices can be reached from all other ones. We denote the white-or-black constraint by $\mathcal{C}$ in this section, and we define it as follows:

$$\mathcal{C} := \bigwedge_{i=1,j=1,i\neq j}^{n} x_i \supset x_j = (x_1 \wedge x_2 \wedge \cdots \wedge x_n) \vee (\neg x_1 \wedge \neg x_2 \wedge \cdots \wedge \neg x_n) \tag{3.3.4}$$

Its negation is called the *Black-and-White* constraint, which is the conjunction of the black clause and the white clause:

$$\neg\mathcal{C} = (\neg x_1 \vee \neg x_2 \vee \cdots \vee \neg x_n) \wedge (x_1 \vee x_2 \vee \cdots \vee x_n) \tag{3.3.5}$$

By the formula $\mathcal{C}$, we state that there is a path from any vertex to any other one. To show that $\mathcal{D}$ is strongly connected we have to show that the logical representation of $\mathcal{D}$, which is $\mathcal{M}$, implies $\mathcal{C}$. Now we can use the fact that SAT solving is the dual of theorem proving. So instead of proving that "$\mathcal{D}$ is strongly connected if $\mathcal{M}$ implies $\mathcal{C}$", we prove that "$\mathcal{D}$ is strongly connected if $\mathcal{M} \wedge \neg\mathcal{C}$ is unsatisfiable". This means that we can use a SAT solver to check a property of a graph.

Motivated by this, we define the following notion: F is a *Black-and-White SAT* problem if and only if F is satisfiable and has exactly two solutions, the white and the black assignments, which implies that $F \wedge \neg\mathcal{C}$ is unsatisfiable. F is a *Black-and-White 2-SAT* problem if and only if F is a 2-SAT and a Black-and-White SAT problem. Lemma 6 in [210] suggests an alternative, more general definition: F is a *Black-and-White SAT* problem if and only if F is satisfiable and has exactly two solutions, A and B such that $A = \neg B$. But we do not use this alternative definition in this section.

First, we need an auxiliary lemma, which states that: there is a path from vertex x_i to x_j if and only if $x_i \supset x_j$ is subsumed by the logical model of the graph.

Lemma 8. *Let $\mathcal{D}$ be a communication graph. Let $\mathcal{M}$ be the 2-SAT representation of $\mathcal{D}$. Then $\mathcal{M}$ implies the formula $x_i \supset x_j$ if and only if there is a path from vertex x_i to x_j in graph $\mathcal{D}$.*

Proof. We know that $\mathcal{M}$ is a special SAT instance where each clause contains exactly one positive and one negative literal, hence, each resolvent of clauses from $\mathcal{M}$ is a binary clause with one positive and one negative literal. The main idea of the proof is that if we have two clauses $\neg v_1 \vee v_2$ and $\neg v_2 \vee v_3$, which means that there is an edge from v_1 to v_2, and from v_2 to v_3, i.e., there is a path in $\mathcal{D}$ from v_1 to v_3, then by resolution we can generate $\neg v_1 \vee v_3$. □

Based on this lemma, we can prove the main theorem of this section which states that the notion of strongly connected graphs and the notion of *Black-and-White 2-SAT* problems are equivalent. This means that the 2-SAT representation of a strongly connected graph is a Black-and-White 2-SAT problem, and the other way round, the directed graph representation of a Black-and-White 2-SAT problem is a strongly connected graph.

Theorem 1. *Let $\mathcal{D}$ be a communication graph. Let $\mathcal{M}$ be the 2-SAT representation of $\mathcal{D}$. Then $\mathcal{M}$ is a Black-and-White 2-SAT problem if and only if the graph $\mathcal{D}$ is strongly connected.*

Proof. Let $\mathcal{D}$ be a communication graph. Let $\mathcal{M}$ be the 2-*SAT* representation of $\mathcal{D}$. We show that both directions hold.

($\Rightarrow$) The main idea of the proof is the following: if a formula is satisfied by all solutions of a *SAT* problem, then it is implied by this SAT problem. We know, that $\mathcal{M}$ has only two solutions, the white and the black assignments, both of them satisfy all $x_i \supset x_j$ shaped formulas, i.e., they are implied by $\mathcal{M}$. From this and from Lemma 8 we obtain that in $\mathcal{D}$ there is a path from any vertex to any other one, i.e., $\mathcal{D}$ is strongly connected.

($\Leftarrow$) The main idea of the proof is the following, since $\mathcal{D}$ is strongly connected we know that there is a path $v_1 \supset v_2, v_2 \supset v_3 \ldots, v_{z-1} \supset v_z, v_z \supset v_1$ which is cyclic and contains all vertices from $\mathcal{D}$, the corresponding clause set is $\{(\neg v_1 \vee v_2), (\neg v_2 \vee v_3), \ldots, (\neg v_{z-1} \vee v_z), (\neg v_z \vee v_1)\}$, which is a subset of $\mathcal{M}$, and which can be satisfied only by the white and the black assignments. From this and since each clause in $\mathcal{M}$ is a binary clause, we obtain, that $\mathcal{M}$ is a *Black-and-White 2-SAT* problem. □

To check whether a directed graph is strongly connected or not, we need linear time [166]. We are going to show that its 2-SAT representation can be solved also in linear time.

Theorem 2. *Let F be a Black-and-White 2-SAT problem. Then we need linear time to show that $F \wedge \neg C$ is unsatisfiable.*

Proof. The main idea of the proof is that any *SAT* solver which uses variable branching and *BCP*, can show that $F \wedge \neg C$ is unsatisfiable by using 1 variable branching and 2 *BCP* steps as follows: Variable branching will result in a unit. Without the lost of generality let us assume that it is a positive one. Then *BCP* will generate other positive units, because the binary clauses in F contain exactly one positive and one negative literal. *BCP* will finally result in a conflict, because $\neg C$ contains the negation of the white assignment. Then the other branch will result in a negative unit. This enables a *BCP* step which will generate negative units, and which will terminate in a conflict because $\neg C$ contains also the negation of the black assignment. Since BCP is a linear time method [193] we need linear time to show that $F \wedge \neg C$ is unsatisfiable. □

Note that *DPLL* algorithm is a suitable choice to solve the above problem because it uses variable branching and *BCP*. The question arises, why should we transform a graph into a *SAT* problem to check a property which can be checked in linear time in both fields? We think that our model is a new link between the two fields which might help to visualize *SAT* problems. This is real problem in case of the 3-*SAT* problem. This thesis suggests that there might be a 3-*SAT* representation of directed graphs which is a Black-and-White 3-*SAT* problem. If one finds that representation, then this could help to visualize 3-*SAT* problems as a directed graph.

3.3.2 Our Model and the Aspvall Model

In this section, we show that the constraint used by our model is more general than the one used by the model of Aspvall et al. First we show some examples. In Figure 3.1, we see a directed graph with 5 vertices. This graph is also a communication graph, since there is no negation sign in the vertices. A directed graph is strongly connected if there is a path between all pairs of vertices. A strongly connected component of a directed graph is a maximal strongly connected subgraph. There are 2 strongly connected components ([1,2,3,5],[4]) in the following graph.

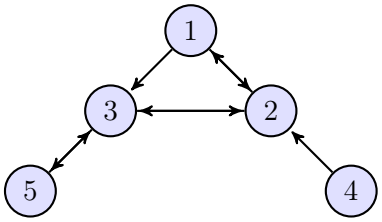


Figure 3.1: A directed graph with 5 vertices

The model of this graph is:

$$\begin{aligned}\mathcal{M} = (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_2 \vee x_3) \wedge \\ (\neg x_3 \vee x_2) \wedge (\neg x_3 \vee x_5) \wedge (\neg x_4 \vee x_2) \wedge (\neg x_5 \vee x_3)\end{aligned} \tag{3.3.6}$$

From $\mathcal{M}$ we can create $D(\mathcal{M})$ by following the construction defined in [8] by Aspvall et al. Construction steps are the following:

i. For each variable x_i, we add two vertices named x_i and $\neg x_i$ to $D(\mathcal{F}_{\mathcal{M}})$.

ii. For each clause $(x_i \vee x_j)$ of $\mathcal{M}$, we add edges $\neg x_i \to x_j$ and $\neg x_j \to x_i$ to $D(\mathcal{M})$.

In Figure 3.2, we see the transformed $D(\mathcal{M})$ graph. Aspvall et al. [8] *(Theorem I.)* shows that $\mathcal{M}$ is satisfiable if and only if in $D(\mathcal{M})$ there is no vertex x_i such that it is in the same strongly connected component as its complement $\neg x_i$. We can be sure that this property holds for our model $\mathcal{M}$ because it is always satisfiable.

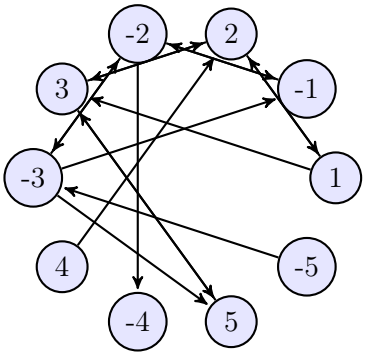


Figure 3.2: $D(\mathcal{F})$, where $\mathcal{F} = \mathcal{M}$

We define the so-called Aspvall kind constraints.

Let $\mathcal{C}_{ASP(i,j)} = (x_i \supset x_j) \wedge (x_i \supset x_j)$, where $1 \leq i,j \leq n$ and $i \neq j$. From this, after simplification, we obtain that $\neg\mathcal{C}_{ASP(i,j)} = (x_i \vee x_j) \wedge (\neg x_j \vee \neg x_i)$. We can add these clauses to the model to check whether there is a directed path from x_i vertex to x_j and vice versa: $D(\mathcal{M} \wedge \neg\mathcal{C}_{ASP(i,j)})$.

We add two constraints to the example shown in Figure 3.1. The first one is $\mathcal{C}_{ASP(1,4)} = (x_1 \supset x_4) \wedge (x_4 \supset x_1)$. Let $\mathcal{F} = \mathcal{M} \wedge \neg\mathcal{C}_{ASP(1,4)}$. So we add these two clauses $(x_1 \vee x_4) \wedge (\neg x_1 \vee \neg x_4)$ and these 4 edges, see the red arrows in Figure 3.3. We see in Figure 3.3, that in graph $\mathcal{D}(\mathcal{F})$ there is no existing vertex x_i which is in the same strong component as its complement $\neg x_i$, hence, $\mathcal{F}$ is still satisfiable.

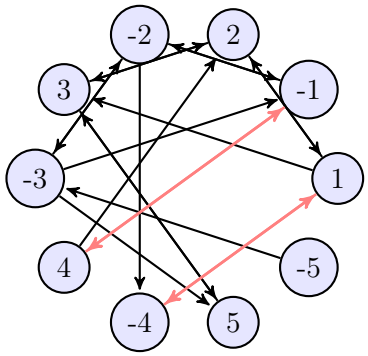


Figure 3.3: $D(\mathcal{F})$, where $\mathcal{F} = \mathcal{M} \wedge \neg\mathcal{C}_{ASP(1,4)}$

Now we add $\mathcal{C}_{ASP(1,5)} = (x_1 \supset x_5) \wedge (x_5 \supset x_1)$. Let $\mathcal{F}' = \mathcal{M} \wedge \neg\mathcal{C}_{ASP(1,5)}$. So we add these two clauses $(x_1 \vee x_5) \wedge (\neg x_1 \vee \neg x_5)$ and these 4 edges, see the red arrows in Figure 3.4. We can see in Figure 3.4. that in the graph $\mathcal{D}(\mathcal{F}')$ there exists a vertex x_i which is in the same strong component as its complement $\neg x_i$, hence, $\mathcal{F}'$ is unsatisfiable.

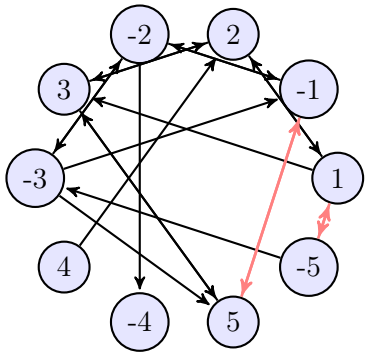


Figure 3.4: $D(\mathcal{F}')$, where $\mathcal{F}' = \mathcal{M} \wedge \neg\mathcal{C}_{ASP(1,5)}$

In our approach C is

$$\begin{aligned}(x_1 \supset x_2) \wedge (x_1 \supset x_3) \wedge (x_1 \supset x_4) \wedge \\ (x_2 \supset x_1) \wedge (x_2 \supset x_3) \wedge (x_2 \supset x_4) \wedge \\ (x_3 \supset x_1) \wedge (x_3 \supset x_2) \wedge (x_3 \supset x_4) \wedge \\ (x_4 \supset x_1) \wedge (x_4 \supset x_2) \wedge (x_4 \supset x_3)\end{aligned} \tag{3.3.7}$$

After eliminating implications:

$$\begin{aligned}(\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_1 \vee x_5) \wedge \\ (\neg x_2 \vee x_1) \wedge (\neg x_2 \vee x_3) \wedge (\neg x_2 \vee x_4) \wedge (\neg x_2 \vee x_5) \wedge \\ (\neg x_3 \vee x_1) \wedge (\neg x_3 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_3 \vee x_5) \wedge \\ (\neg x_4 \vee x_1) \wedge (\neg x_4 \vee x_2) \wedge (\neg x_4 \vee x_3) \wedge (\neg x_4 \vee x_5) \wedge \\ (\neg x_5 \vee x_1) \wedge (\neg x_5 \vee x_2) \wedge (\neg x_5 \vee x_3) \wedge (\neg x_5 \vee x_4)\end{aligned} \tag{3.3.8}$$

After simplification, $\neg C$ is the following:

$$(x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee \neg x_5) \tag{3.3.9}$$

For any i and j we have that $\neg\mathcal{C}_{ASP(i,j)}$ implies $\neg\mathcal{C}$, thus $\neg\mathcal{C}_{ASP(i,j)}$ subsumes $\neg\mathcal{C}$. This means that $\neg\mathcal{C}_{ASP(i,j)}$ defines a stronger constraint to the model, thus $\neg\mathcal{C}$ defines a more general constraint than $\neg\mathcal{C}_{ASP(i,j)}$ to check whether a graph is strongly connected or not.

3.3.3 Connectivity Test by *SAT* Representation

We primarily focus on randomly deployed sensor networks. A sensor network can be either a homogeneous or heterogeneous group of nodes. In the case of homogeneous sensors, the graph is symmetric and it is also equivalent to a simple undirected graph.

If we have a *WSN*, then we can redefine a communication graph as follows: we say that $\mathcal{D}$ is a communication graph if and only if the elements of $\mathcal{V}$, the vertices, represent the sensor nodes of the *WSN*, and elements of $\mathcal{E}$, the edges, represent a one way communication between two nodes. We intend to examine which sensors can communicate with which ones, thus we can create the communication graph and its 2-*SAT* representation. In this approach, it can be checked fairly quickly in a given state of a randomly distributed sensor network *(assuming all sensors are awaken)* whether it can be ensured that each sensor can communicate with every other one [209].

An Example Let our model a randomly distributed heterogeneous sensor network which has 10 nodes. The communication graph can be seen in Figure 3.5. For representing the input clause set for a *SAT* solver, we use the *DIMACS CNF* format. The generated *DIMACS CNF* format is the following:

```
p cnf 10 20
c model
```

```
-1 6 0
-6 1 0
...
-10 8 0
-9 5 0
c constraint
1 2 3 4 5 6 7 8 9 10 0
-1 -2 -3 -4 -5 -6 -7 -8 -9 -10 0
```

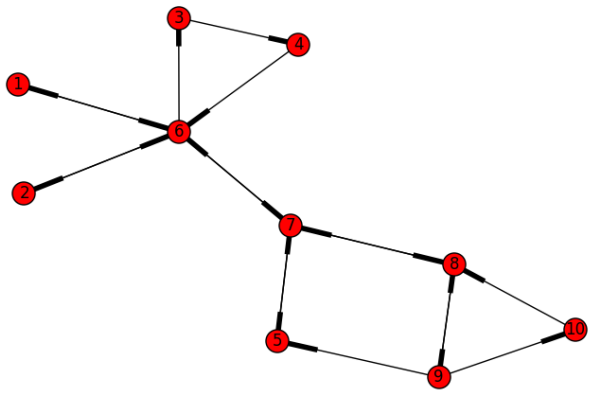


Figure 3.5: Heterogeneous sensor nodes with their communication graph

Our main goal is to examine the produced *DIMACS CNF* file with `MiniSat 2.2.0`[3], which is a complete SAT solver, which returns *UNSAT*. Thus the model fulfills the requirements, namely the communication is ensured between any two nodes. In this example, the result is *UNSAT*, thus the represented communication graph is strongly connected.

This model can also give a more detailed analysis. For example, if we take the sensors out of the model one by one, and with the remaining ones we test the communication problem again between any two nodes. If the solver returns satisfiable (*SAT*) for the reduced model, then the currently removed node is extremely important to the sensor network, as its removal breaks the requirement of unhindered communication between any two sensors. The significance of this in the graph theory is that the removal of this node makes the graph not strongly connected anymore. In our example, the removal of nodes `3`, `4`, `6`, `7`, `8` and `9` would return with *SAT*, thus the malfunction of those sensors means the communication in that sensor network which is inadequate.

Generally true *(see in Section 4.3)*, that if the graph is strongly connected, then a

[3] http://minisat.se

DPLL-based *SAT* solver returns the following values.

- number of solutions : 0
- number of conflicts : 2
- number of decisions : 1
- number of unit propagations : $u = 2m$, where m the number of clauses *(number of vertices)*

These results also show that the *Black-and-White 2-SAT* problem with the *Black-and-White* constraint can be solved in linear time since the number of decisions is 1 and the number of unit propagations is $2m$.

3.4 Conclusions

In the first section, we have introduced two *SAT* instance generators, $WnDGen1$ and $WnDGen$, which generate highly structured (hard combinatorical) *SAT* instances, which have very interesting properties. We have shown that around this threshold there are *SAT* instances, which are difficult for state-of-the-art *SAT* solvers, i.e., they are good for testing *SAT* solvers. In the second section, we have introduced a *SAT* representation that can be used for modelling directed graphs. The only restriction is that the names of the vertices should be boolean variables. This model, which is a 2-*SAT* problem, makes it possible to check whether a graph is strongly connected by adding two clauses, the black and the white ones, to the model and asking a SAT solver whether this formula is unsatisfiable. The two constraint clauses are, which state that it is not true that there is a path from any vertex to any other one. This constraint is more general than the one used by Aspvall et al. We have shown that the representation of a strongly connected graph is a *Black-and-White 2-SAT* problem. The *Black-and-White SAT* problem appears also as a special case of weakly nondecisive *SAT* problems, see Lemma 6. in [210]. This suggests two things: the *Black-and-White SAT* problem could be an interesting problem in general, and since there are weakly nondecisive 3-*SAT* problems, which are also *Black-and-White*, there might be a 3-*SAT* representation of directed graphs.

4 Part II

In this chapter first, we present two sequential *SAT* solvers, and show how to use them to solve *WSN* based *SAT* problems. At the end of this chapter, we show how to develop techniques for using distributed computing resources to efficiently solve instances of the propositional satisfiability problem. We present a parallel *SAT* solver `CCGrid`, which runs on the MTA `SZTAKI` Grid using `BOINC`.

4.1 Introduction

Modern *SAT* solvers use a number of previously described techniques *(backjumping, restarts, lemma learning, conflict-driven clause learning, machine learning, background theories and many other - Bohm's, MOMS, Jeroslow-Wang, (R)DLCS, (R)DLIS, LEFV - heuristics, see [22])*, but most of them are based on *DPLL* algorithm. *DPLL* abstract algorithm is a complete, backtracking-based search algorithm for deciding the satisfiability problem. Beyond the backtracking, it uses the following rules at each step: unit propagation (*UP*) and pure literal elimination (*PLE*). This pure literal elimination is also called pure literal rule or monotone literal rule. A clause with only one literal is called a unit clause and the literal x is pure in a `CNF` formula F if $\neg x$ does not occur in F. A simple *DPLL*-based *SAT* solver spends most of its runtime in *UP* and *PLE* methods. The direct implementation of UP and *PLE* takes quadratic polynomial time in the total size of the formula to check. The UP has a linear time implementation given by [193]. In 2005 Johannsen [87] proved that the `PLE` is P-complete in general and NL-complete for 2-SAT.

4.2 The `CSFLOC`

In this section we describe, the `CSFLOC` algorithm [205], i.e., the `C`ounting `S`ubsumed `F`ull- `L`ength `O`rdered `C`lauses algorithm. As its name shows, `CSFLOC` counts full-length clauses, and the clauses are ordered, which means that we use a fixed order of boolean variables, which are presented in the input problem. This order gives us an index for each variable. The index of a literal is the index of its variable. `CSFLOC` is the successor of the `Optimized CCC` algorithm [207], `CCC` algorithm [207]. By studying `Optimized CCC`, we observed that its full-length clause counter can be increased on its last 1 bit in the best case. We proved that this observation is generally true for `Optimized CCC` [205]. The new algorithm, `CSFLOC`, uses this result. It uses also a data structure in which the

clauses are ordered by the index of their last literal according to the fixed order of boolean variables. These two improvements result in a faster algorithm which can compete with a state-of-the-art *SAT* solver on problems with lots of clauses, like *Black-and-White 2-SAT* problems and weakly nondecisive *SAT* problems.

The `CSFLOC` algorithm and its predecessors are motivated by the following idea: We know that there are 2^n different full-length clauses, where n is the number of boolean variables in the input clause set. A *SAT* problem is unsatisfiable if it subsumes all full-length clauses. If a *SAT* problem subsumes fewer then it is satisfiable. These observations can be derived from the Clear Clause View from [103]. This means that by counting the subsumed full-length clauses we can decide satisfiability.

So, by counting the subsumed full-length clauses, we can tell how many models the input problem has. If this number is zero, then the problem is unsatisfiable, otherwise it is satisfiable. This idea is not new. This idea comes from $\#SAT$.

An interesting question is how many models a *SAT* problem instance has. This problem is known as the $\#SAT$ problem. In other words, by $\#SAT$ we mean the problem of counting the models of a *SAT* instance [79]. The most popular way to solve this problem is to use a variant of the *DPLL* method which does not stop if it finds a model but keeps exploring the search space. One of the first examples is *CDP* [25].

Another way to count the models is to use the inclusion-exclusion principle. It is well known that this principle can solve the $\#SAT$ problem [84, 121, 17].

Instead of using the inclusion-exclusion principle one can use our new algorithm, the `Counting Subsumed Full-Length Ordered Clauses` algorithm, for short `CSFLOC`. It counts the subsumed full-length clauses, like the inclusion-exclusion principle, but in an iterative way.

We know that `CSFLOC` is the successor of the `CCC` algorithm, which is a very simple algorithm, hence, it is easy to show that it is sound and complete [205].

`CCC` works as follows: It sets its counter to be 0. It converts 0 to a full-length clause, called C, which is the one with only negative literals. In general, C is created from the counter in the following way: Each bit of the counter is represented by a literal, the literal is positive if the corresponding bit is 1, negative otherwise. It checks whether this full-length clause is subsumed by the input problem. This means that it looks for a clause D from the input SAT problem that is a subset of C. If such a clause D is found, then it adds 1 to the counter and repeats the process until it finds a full-length clause which is not subsumed, which means that the input is satisfiable, or all possible full-length clauses are visited, i.e., the input is unsatisfiable.

It is easy to see that this algorithm stops for any *SAT* problem and can decide whether the *SAT* instance is satisfiable or not. This means that this simple algorithm is sound and complete, but in the case of an unsatisfiable *SAT* instance, it visits all the 2^n full-length clauses.

The next version is called `Optimized CCC`, which uses the observation that a clause

subsumes 2^{n-i} consecutive ordered full-length clauses, where i is the index of its last literal according to an order of the boolean variables of the clauses. This means that if we find a clause D which subsumes C, then we can increase the counter by 2^{n-i} instead of 1. So we do not need to visit all full-length clause to decide satisfiability. The `Optimized CCC` always selects that D in the loop which grants the biggest step, i.e., where the number 2^{n-i} is the greatest. This optimization does not affect the soundness and the completeness of the algorithm [205].

The next version of this algorithm family is `CSFLOC`. Here we use the observation that if we always use the best D in the loop, as in the `Optimized CCC`, then the biggest possible step is 2^{n-j}, where j is the index of the last 1 bit in the counter. Note, the order of bits in the counter should correspond to the order of the boolean variables, which is used to order the clauses. Note, that this bit always comes from the previous run of the loop. It is so because if there were an even better D in the input clause set, then the algorithm would select that D even in the previous loop and setting a bit to 1 which has smaller index than j. This means that we do not have to scan the whole input clause set to find the best D, we have to consider only those clauses where the index of the last literal is greater than or equal to j. This optimization does not affect the soundness and the completeness of the algorithm [205].

Counting full-length clauses is not a new idea. Already Iwama used this idea in 1989 [84]. Later Lozinskii introduced an approximation algorithm for counting models [121]. They count full-length clauses following the inclusion-exclusion principle more closely. This means that they also decrement the counter, see Algorithm 3 in [17]. In the case of the `CCC` algorithm family we always increment the counter. So we can use the above observation about the biggest possible step, while the classical algorithms cannot do so.

Full-length clauses are called also maximal clauses in the literature. A good overview of these papers can be found in [11]. In this paper, Andrei introduces an inverse resolution to generate maximal clauses. He also overviews the field of maximal clause counting. The two basic papers in this field are Tanaka's paper [172] and Dubois' paper [53]. Neither of them uses the term maximal clause but they use techniques which build bigger and bigger clauses, i.e., eventually they build maximal clauses, to have independent clauses (two clauses are independent if they do not subsume the same maximal clause). In this field, we could not find a similar algorithm to `CSFLOC`. Now we introduce the `CSFLOC` formally.

In this algorithm, $IndexOfLastLiteral(C)$ gives back the index of last literal of C according to the variable order. We assume that this order goes from 1 to n, where n is the number of variables in S, which is the input problem.

We assume that *count* is an $n+1$-bit-wide non-negative integer, where the index of the highest bit is zero. $FullLengthClauseRepresentationOf(count)$ converts *count* to the full-length clause, where the lowest bit corresponds to the last boolean variable, and it is a negative literal, if the bet is 0, otherwise it is positive. The highest bit is not converted.

$IndexOfLastPositiveLiteralOrOne(C)$ gives back the index of the last positive literal

Algorithm 1 `CSFLOC`(S)

Require: S is a non-empty list of ordered clauses with variable indexing function I.

Ensure: If S is satisfiable it returns a model for S, otherwise returns the empty set.

1: $n :=$ number of variables in S;
2: $S[i] := \{C | C \in S \wedge \text{IndexOfLastLiteral}(C) = i\}$, where $i = 1..n$;
3: $count := 0$;
4: **while** $count < 2^n$ **do**
5: $increment := 0$;
6: $C :=$FullLengthClauseRepresentationOf($count$);
7: **for** $j :=$IndexOfLastPositiveLiteralOrOne(C); $j <= n$; $j := j + 1$ **do**
8: **if** $\exists D \in S[j]$ such that D subsumes C **then**
9: $increment := 2^{n-j}$;
10: $j := n + 1$;
11: **end if**
12: **end for**
13: **if** $increment = 0$ **then**
14: return $\overline{C}$
15: **else**
16: $count := count + increment$;
17: **end if**
18: **end while**
19: return {};

in C according to the variable order. If there is no positive literal in C, then it returns 1. Note, that we assume that our variable order starts from 1.

The observation is that the biggest possible step is 2^{n-j}, where j is the index of the last 1 bit in the counter. Note, that this bit always comes from the previous run of the loop. It is so because if there were an even better D in the input clause set, then the algorithm would select that D even in the previous loop and setting a bit to 1 which has smaller index than j. This optimization does not affect the soundness and the completeness of the algorithm.

Implementation

We have implemented the texttttCCC in *Java* and *C#*. The implementation can be downloaded from `http://fmv.ektf.hu/files/CCCv1.0.zip`.

We have also implemented the `Optimized CCC` in `ANSI C`. The implementation can be downloaded from `http://fmv.ektf.hu/files/opt_ccc.c`. The main idea of the implementation is to use arrays of integers for the bit representation of clauses. Since we use integers we can use fast bit operations to implement the methods of a clause.

We have implemented the CSFLOC in Java, see http://fmv.ektf.hu/tools.html. For the tests we used this version: http://fmv.ektf.hu/files/CSFLOC6.java.

4.2.1 Test Results

We tested the ANSI C version of the Optimized CCC and CSFLOC. The tests were done on iMac macOS Sierra (CPU: 2,5GHz Intel Core i5, Memory: 4GB 1333MHz DDR3). We tested our algorithm against Glucose[1] 3.0, which is a widely used standard *SAT* solver. We used Glucose 3.0 with simplification mode turned on. The *SAT* instances were downloaded from the *SATLIB* Benchmark Problems [2] page. The rest is generated by the *WnDGen SAT* problem generator [202].

For each instance we used a timeout of 900 seconds. The results are presented in table format. The first column is the name of the tested *SAT* instance. In the 2nd to 3rd columns, we show the number of variables (n), the number of clauses (m). The last three columns are *CPU* times in seconds for Glucose 3.0, Optimized CCC and CSFLOC.

One of our goals was to countercheck the completeness of CSFLOC on a lot of problems. Table 4.1 shows that we have run it for all uf20*, uf50*, and uuf50* instances from *SATLIB* Benchmark Problems. For all the 3000 instances it returned right result. The same is true for all the other tests we performed.

Table 4.1 shows that random *SAT* instance, like uf50* and uuf50*, are difficult for CSFLOC, but very easy for Glucose. We can also see that the CSFLOC outperforms Optimized CCC. We can see on lines hole 6-9 that on pigeon hole problems, where we place $n+1$ pigeons in n holes without placing 2 pigeons in the same hole, the CSFLOC can compete with the Glucose if the number of variables is relatively small. It is so because in pigeon hole problems there are lots of short clauses, which help CSFLOC to do big steps.

	n	**m**	Glucose	CSFLOC	OptCCC
uf20*	20	91	0,0042s	0,0046s	0,0043s
uf50*	50	218	0,0044s	1,0477s	5,1632s
uuf50*	50	218	0,0041s	5,4807s	26,2165s
hole6	42	133	0,0085s	0,0061s	0,0284s
hole7	56	204	0,0920s	0,0758s	0,5499s
hole8	72	297	1,1338s	1,1734s	12,1701s
hole9	90	415	13,1981s	19,7952s	TIME OUT

Table 4.1: Runtimes on problems from *SATLIB* Benchmark Problems, uf* are *SAT*, others are *UNSAT*

Since the CSFLOC uses a variable ordering, a natural question arises, whether a "better" variable ordering results in a better runtime result? To answer this question we used

[1] http://www.labri.fr/perso/lsimon/glucose/

[2] http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

the variable clustering tool written by Prof. Tudor Jebelean [86]. The tool is available at `http://aries.ektf.hu/~gkusper/Clustering-8-Jul-2010.tar.gz`. Our experience shows, see Table 4.2, that if we cluster the variables of SAT problems then `CSFLOC` runs faster. Especially it can solve random *SAT* problems with 75 or 100 variables, which was otherwise very difficult for `CSFLOC`.

Instances	**n**	**m**	**clustering factor**	`CSFLOC`
`uf20*`	20	91	no clustering	0,0046 s
`uf20*`	20	91	clusters of 2 vars	0.0033s
`uf20*`	20	91	clusters of 3 vars	0.0031s
`uf50*`	50	218	no clustering	1,0477s
`uf50*`	50	218	clusters of 2 vars	0.2161s
`uf50*`	50	218	clusters of 3 vars	0.7917s
`uf75*`	75	325	no clustering	TIME OUT
`uf75*`	75	325	clusters of 2 vars	147,7269s
`uf75*`	75	325	clusters of 3 vars	129,1973s
`uf100*`	100	430	no clustering	TIME OUT
`uf100*`	100	430	clusters of 2 vars	TIME OUT
`uf100*`	100	430	clusters of 3 vars	25,0288s

Table 4.2: Runtimes on problems after clustering, all instances are *SAT*

4.2.2 IoT Inspired Test Results

In section 3.3, a method was introduced to translate a directed graph into a *SAT* problem. The motivation was to check whether the communication graph of a *WSN* is strongly connected or not. We showed that if a directed graph is strongly connected, then the generated *SAT* instance is a Black-and-White *SAT* problem. A *Black-and-White SAT* problem has only two solutions, the white assignment in which each variable is true, and the black assignment in which each variable is false.

Now we use this result to generate test problems. We generated *WSN*s with 100 sensors with different density and translated them into *Black-and-White 2-SAT* problems. We also added the black and the white assignments to the tested *SAT* instances to make the strongly connected ones to be unsatisfiable. The less dense *WSNs* are not strongly connected, i.e., the resulting *SAT* problems are satisfiable. The more dense *WSNs* are strongly connected, i.e., the corresponding *SAT* instances are unsatisfiable.

We tested `Glucose` and `CSFLOC` on these instances. Table 4.3 shows the test results. The name of the instance indicates whether it is satisfiable *SAT*, or unsatisfiable *UNSAT*. We can see that `Glucose` is around 10 times faster than `CSFLOC`.

We also generated *WSNs* with more sensors: 10-10 instances with 200, 500, 1000, 2000 sensors and with different density. We used the same method described in the previous test case, i.e., we translated the *WSN*s into *Black-and-White 2-SAT* problems. Table 4.4 shows

	n	m	Glucose	CSFLOC
WSN_100_SAT_1.cnf	100	676	0,0017s	0,0181s
WSN_100_SAT_2.cnf	100	852	0,0021s	0,0202s
WSN_100_SAT_3.cnf	100	837	0,0022s	0,0201s
WSN_100_UNSAT_1.cnf	100	1464	0,0026s	0,0241s
WSN_100_UNSAT_2.cnf	100	1458	0,0030s	0,0230s
WSN_100_UNSAT_3.cnf	100	2263	0,0037s	0,0271s
WSN_100_UNSAT_4.cnf	100	4631	0,0060s	0,0370s
WSN_100_UNSAT_5.cnf	100	6172	0,0093s	0,0461s
WSN_100_UNSAT_6.cnf	100	8134	0,0047s	0,0481s
WSN_100_UNSAT_7.cnf	100	842	0,0021s	0,0240s

Table 4.3: Runtimes on *Black-and-White 2-SAT* problems

the average test results. Here the number of clauses, i.e., column **m** is an average number. We can see that from 1000 sensors the CSFLOC algorithm is better to check whether the communication graph of a *WSN* is strongly connected or not. This result was a big surprise for us because these SAT problems have the same number of variables as the number of sensors, see column **n**. It shows that a random *SAT* instance is very difficult for the CSFLOC already with 100 variables but not a *Black-and-White 2-SAT* problem.

	n	m	Glucose	CSFLOC
WSN_100*	100	2732,9	0,0038s	0,0317s
WSN_200*	200	5171,6	0,0108s	0,0494s
WSN_500*	500	50310,0	0,1499s	0,1652s
WSN_1000*	1000	271103,6	2,4919s	0,5336s
WSN_2000*	2000	1006531,6	20,1476s	3,7704s

Table 4.4: Average runtimes on *Black-and-White 2-SAT* problems

The *Black-and-White SAT* problem is a special weakly nondecisive *SAT* (*WnD*) problem, see section 3.3. *WnD* problems can be generated by our tool, called WnDGen tool, available here: `http://fmv.ektf.hu/tools.html`. We recall, that these problems have an interesting property, they have only two solutions, and the two solutions are the opposite of each other, as in the case of *Black-and-White 2-SAT* problems.

It seems that CSFLOC is very good at solving *WnD* problems generated by the WnDGen *SAT* problem generator, see Table 4.5.

To create the *WnD* problem instances we used the "-unsat" switch of WndGen tool, which adds the negation of the two solutions of the *WnD* problems to the instance, making it unsatisfiable. This shows that overconstrained problems with few variables, like *WnD* problems, are easier for the CSFLOC, than general *SAT* problems.

Instances	n	m	Glucose	CSFLOC
WndGen_unsat_10_3.cnf	10	722	0,0017s	0,0281s
WndGen_unsat_10_4.cnf	10	1682	0,0112s	0,0242s
WndGen_unsat_10_5.cnf	10	2522	0,0276s	0,0321s
WndGen_unsat_10_6.cnf	10	2522	0,0316s	0,0532s
WndGen_unsat_15_3.cnf	15	2732	0,0027s	0,0281s
WndGen_unsat_15_4.cnf	15	10922	0,2088s	0,0893s
WndGen_unsat_15_5.cnf	15	30032	1,9055s	0,1282s
WndGen_unsat_15_6.cnf	15	60062	9,3856s	0,2062s
WndGen_unsat_15_7.cnf	15	90092	25,2388s	0,2991s
WndGen_unsat_15_8.cnf	15	102962	38,8714s	0,4343s
WndGen_unsat_15_9.cnf	15	90092	36,3287s	3,1751s
WndGen_unsat_20_3.cnf	20	6842	0,0063s	0,0532s
WndGen_unsat_20_4.cnf	20	38762	1,8473s	0,1271s
WndGen_unsat_20_5.cnf	20	155042	37,5251s	0,3022s
WndGen_unsat_20_6.cnf	20	465122	TIME OUT	0,7083s
WndGen_unsat_20_7.cnf	20	1085282	TIME OUT	3,5827s
WndGen_unsat_20_8.cnf	20	2015522	TIME OUT	8,7173s
WndGen_unsat_20_9.cnf	20	3023282	TIME OUT	19,7737s

Table 4.5: Runtimes on WnD problems *(all instances are UNSAT)*

4.2.3 How to Create a Parallel Version?

From CSFLOC we can create naturally a parallel algorithm. Assume we have q clients, then the j.-th CSFLOC instance has to count from $j \cdot 2^{n/q}$ till $(j+1) \cdot 2^{n/q-1}$, where $j = 0 \dots q-1$ and n is the number of variables in the input SAT problem.

The algorithm is so simple that there is no necessary communication between the clients, except the signal that one of the clients has found the solution, which stops also the other clients.

Each node has to maintain only a counter, they can share the input SAT problem because it does not change during the execution. This means that the memory usage is minimal.

4.2.4 Future Work

One could find a closer connection between the inclusion-exclusion principle and CSFLOC. Let us assume, we have the following situation:

An example Let us have 3 variables, $\{a, b, c\}$, i.e., $n = 3$. Let $I(a) = 1, I(b) = 2, I(c) = 3$. Let *count* be 0, so $D = \{\overline{a}, \overline{b}, \overline{c}\}$. Let the input clause set be $\{C_1, C_2\}$, where $C_1 = \{\overline{a}, \overline{b}\}$,

$C_2 = \{\overline{a}, \overline{c}\}$. We can see that both C_1 and C_2 subsumes D. By the `CSFLOC` algorithm we have to increase *count* only by 2^{3-2} because the minimum of maximum indices among C_1 is $I(b) = 2$.

Now $count = 2$ and $D = \{\overline{a}, b, \overline{c}\}$. This is subsumed only by C_2. Its last variable is c and $I(c) = 3$. So we can increase *count* by 2^{3-3}. Now $count = 3$ and $D = \{\overline{a}, b, c\}$. It is not subsumed, so its negation is a solution.

In the above example, we can see that we find the same clause, C_1, in the first and also in the second loop, but we used it only in the second one. So it seems that in the first loop we could use both C_1 and C_2. We believe that the inclusion-exclusion principle could help us to find a better solution here.

A clever data structure could help to find directly those clauses from the input *SAT* problem which subsumes the counter. We have no suggestion here.

4.3 The `BaW 1.0`

In the previous chapter, we proved that if a directed graph is strongly connected, then the generated 2-*SAT* problem is a Black-and-White 2-SAT problem, which has two solutions: where each variable is true (the white assignment), and where each variable is false (the black one). We also proved that these problems could be solved in linear time. In this section, we present a *DPLL*-based problem specific *SAT* solver called `BaW 1.0`. It is problem specific because `BaW 1.0` is not a complete *SAT* solver, it is only validated for our special *Black-and-White 2-SAT* problem-based benchmarks. First, we show how to solve the *Black-and-White 2-SAT* problems in linear time and how to use this solver for effective strong connectivity testing in large and sparse directed graphs. After that, we present our results on these benchmarks which show that remarkable improvements have taken place in the number of solved instances as well as in the computation time compared to existing modern state-of-the-art *SAT* solvers *(`CSFLOC 8` see in Chapter 4.2, `Glucose 3.0`[3])*.

Introduction

In this section, we present algorithms that check the strong connectivity of a directed graph in $O(m+n)$ time in the worst case where m is the number of clauses *(vertices)* and n is the number of literals *(edges)*. We know that this result has been already achieved by [51, 75, 166]. In the case of directed graphs, they achieved $O(m+n)$ time by applying graph theoretical methods. With classic *SAT* solving methods this theoretical minimum time is inaccessible. A general *DPLL*-based *SAT* solvers in 2-*SAT* problems are quadratic time complexity. In this section, we present a linear time *SAT* solver *(called* `BaW 1.0`*)* for the *Black-and-White 2-SAT* problems.

[3] `https://www.labri.fr/perso/lsimon/glucose`

4.3.1 Preliminaries

We are primarily going to design the `BaW 1.0` solver for communication networks. Some additions to the model described above *(see in Chapter 3.3.2)*. The *communication clause* ($ComC$) is a clause with two literals, a positive and negative literal on two different variables *(e.g. -lit$_x$ lit$_y$)*. The *constraint clauses* are $ConC_+$ and $ConC_-$. The $ConC_+$ is a full-length clause with only positive literals *(white clause)*. The $ConC_-$ is a full-length clause with only negative literals *(black clause)*. The valid clauses in our benchmark are instances of $ComC$, and the clauses $ConC_-$, $ConC_+$. If the formula contains only valid clauses, we call it a *network-based communication* problem. For such a type of problem, we have called *source* the negative literals and *sink* the positive ones. Additional notations for algorithms: S, c and n denote a clause set, a clause in S, and the number of variables in S, respectively.

4.3.2 Basic Concepts of `BaW 1.0`

In this section, we showed on *Black-and-White 2-SAT* problems that if the graph is strongly connected (*Black-and-White 2-SAT* problem), then a pure *DPLL*-based *SAT* solver returns the following values:

- number of solutions : 0 ,
- number of conflicts : 2 ,
- number of decisions : 1 ,
- number of unit propagations : $u = 2m$, where m the number of clauses *(number of vertices)*.

These results also showed that the *Black-and-White 2-SAT* problem with the *Black-and-White* constraint can be solved in linear time since the number of decisions is 1, and the number of unit propagations is $2m$.

Our solution consists of three main components: parsing and preprocessing, splitting and the main method of `BaW 1.0`.

Parsing and Preprocessing

During the parsing process, we check the correctness of clauses then examine the occurrence of literals. The Algorithm 2 first looks for non-valid clauses. If it finds a non-valid clause, it stops and returns *false*. In this case, the clause set (S) does not match `BaW 1.0` solver input, it is not a network-based communication problem. From the valid $ComC$ clauses we collect the literals and we defined five data structures *(lists and dictionaries)*:

- $contNegLiteralsList$,
- $contPozLiteralsList$,

- *literalList*,
- *oneHopChildrensDict*,
- *oneHopParentsDict*.

If the literal is not an element in the *literalSet*, we add it. Since we are collecting positive and negative literals, so if all literals occur in S, then *literalSet* size is $2n$. For effective linear time *PLE* we store the neighbors (*oneHopChildrensDict*, *oneHopParentsDict*) within 1-hop environment of each vertex *(literal)*. One of the ways of a linear time *UP* is to store for each variable the list of clauses in which each literal is contained. For this purpose we build two lists (*contNegLiteralsList*, *contPozLiteralsList*) of clauses in which each negative and positive literal is contained. In the *contNegLiteralsList* the negative literals are stored and in the *contPozLiteralsList* the positive ones. In Algorithm 2 we also show that these data structures can be built in time linear in the size of the formula. If the size of *literalList* is less than $2n$, the algorithm stops and returns *true*.

Since in our communication model we represent the communication by logical implication, a binary clause represents the possibility of sending a message. If sensor x_1 can send message to sensor x_2 , the model is $(x_1 \supset x_2)$, and the binary clause is $(\neg x_1 \vee x_2)$. In our communication graph, the negative literals represent outgoing edges and positive ones are incoming edges. If the *literalList* size is less than $2n$, then it means that there is at least one vertex in which there is no outgoing or/and incoming edge, which means that the communication graph is not strongly connected, i.e., if the result of parsing and preprocessing is *true*, then S is satisfiable.

Splitting

A general splitting rule is to select a variable v which has not been assigned a truth-value. Assign one truth value t to it, simplify F to F' and call `DPLL`(F'). There are many branching heuristics to provide help with choosing the branching literal [22] *(Chapter 7)*, [128]. The general aim is to minimize the number of decision steps while imposing a minimal computational overhead [94]. Of course, there is a cost for producing and maintaining of splitting and in the worst case these also can increase the runtime.

The logically splitting belongs to preprocessing. For example, in our case, when we check the strong connectivity of a graph, we must consider the whole search space irrespective of what the first splitting literal was.

By default the Algorithm 3 finds the most occurring source or sink type literals, which determines the first branch position (P). But we also have the option of starting a search from a dedicated source or sink.

Algorithm 2 `Parsing and Preprocessing`

Require: F is a *DIMACS CNF* file.

function PARSING_AND_PREPROCESSING(F)
    $definedDataStructures \leftarrow Null$
    $n \leftarrow 0$
    while not EOF **do**
        $c \leftarrow$ readLine
        $n \leftarrow n + 1$
        if c is not ($ComC$ or $ConC_+$ or $ConC_-$) **then**
            return $False$
        else
            if c is $ComC$ **then** or lit_1, $\neg lit_2$
                for each $l \in c$ **do**
                    if $l \notin literalList$ **then**
                        $literalList.add(l)$
                    end if
                    if $\neg lit_1$ and lit_2 **then**
                        $oneHopChildrensDict[\neg lit_1].add(lit_2)$
                        $oneHopParentsDict[lit_2].add(lit_1)$
                        $contNegLiteralsList[n].add(lit_1)$
                        $contPozLiteralsList[n].add(lit_2)$
                    else if lit_1 and $\neg lit_2$ **then**
                        $oneHopChildrensDict[\neg lit_2].add(lit_1)$
                        $oneHopParentsDict[lit_1].add(lit_2)$
                        $contPozLiteralsList[n].add(lit_1)$
                        $contNegLiteralsList[n].add(lit_2)$
                    end if
                end for
            end if
        end if
    end while
    if $length(literalList) \neq 2n$ **then**
        return $True$
    else
        return $False$
    end if
end function

Algorithm 3 `Decide of the First Branch`

function DECIDEFIRSTBRANCH(F)
    $P \leftarrow$ (the most common literal from the **source** or **sink** type literals) **or** (a dedicated **source** or **sink**)
    return P
end function

Main Method of `BaW 1.0`

In Algorithm 4, we can see the main method of `BaW 1.0` Solver. Variable d donates the number of decisions. The $Units$ and $isThereWereUnits$ sets we use to store the current and the already existing units. There are two differences compared to a classic `DPLL`: to use a Modified Unit Propagation (`MUP`) and Decision Detection (`DD`). `MUP` will be discussed in more details later. In our model the number of decision is 1. If, besides the first decision, we are forced to make another decision, then it means the problem is satisfiable. In this case it means that the `MUP` chain is broken, the represented graph is not strongly connected.

Algorithm 4 `The BaW method`

```
1:  function BaW(S)
2:      S ← Modified_Unit_Propagation(S)
3:      if S is a consistent set of literals then
4:          return True
5:      end if
6:      if S contains an empty clause then
7:          return False
8:      end if
9:      P ← DecideFirstBranch(F) or arbitrary P
10:     d ← d + 1
11:     if d > 1 then
12:         return True
13:     end if
14:     Units ← Units ∪ P
15:     if BaW(S ∪ P) then
16:         return True
17:     else
18:         Units ← Units ∪ ¬P
19:         return BaW(S ∪ ¬P)
20:     end if
21: end function
```

In Algorithm 5, we can see the Modified Unit Propagation method of `BaW 1.0`. Note, that this method also includes the pure literal elimination. As mentioned earlier, for the linear time `MUP`, the positions of each literal are stored ($contNegLiteralsList$, $contPoz - LiteralsList$). The `MUP` works differently, depending on whether P is negative or positive. If $P < 0$ it means that the literal is source type. In both cases, we have to delete every clause in S which contains P and all occurrences in S of $\neg P$. In order to find new units, we do not have to look through the whole remaining formula. In the negative case, the P parents ($oneHopParentsDict$) will be the newer units and in the positive case, it will be the children ($oneHopChildrenDict$). After we have determined the new units and deleted every clause in S which contains P and every occurrence in S of $\neg P$, we add P to the $isThereWereUnits$ and delete it from the $Units$.

Algorithm 5 `Modified Unit Propagation`

```
1: function MODIFIED_UNIT_PROPAGATION (S)
2:     for each P ∈ Units do
3:         delete every clause in S containing P
4:         delete every occurrence in S of ¬P
5:         if P < 0 then
6:             if P ∉ isThereWereUnits then
7:                 Units ← Units ∪ oneHopParentsDict[P]
8:             end if
9:             isThereWereUnits ← isThereWereUnits of P
10:        else
11:            if P ∉ isThereWereUnits then
12:                Units ← Units ∪ oneHopChildrenDict[P]
13:            end if
14:            isThereWereUnits ← isThereWereUnits ∪ P
15:        end if
16:        Units ← Units − P
17:    end for
18: end function
```

4.3.3 Test Results

The following describes the results of some connectivity tests. Our tests were done on Intel quadcore machine with 8 GB of RAM running at 2.66 GHz. For testing, we used the `CSLFOC 8` , `Glucose 3.0` and `BaW 1.0` solvers. We note, in these studies the `BaW 1.0` was run without `DecideFirstBranch` method.

Figure 1 shows the direct implementation of *DPLL* and the `BaW 1.0` comparison. Each instance was a complete graph. The x-axis shows the amount of n and m, where n, m are the number of literals *(vertices)* and the number of clauses *(edges)*, respectively. The y-axis shows the average running times. It can be seen from the run times that the direct implementation of *DPLL* is polynomial while `BaW 1.0` is linear.

In section 4.2, we showed that the `CSFLOC` algorithm on *Black-and-White 2-SAT* problems for large network problems is around 5 times faster than `Glucose`. Therefore, in further tests, the `BaW` is compared to `CSFLOC` only.

Figure 2 shows a comparison of the runtime of `CSFLOC` and `BaW 1.0` with `Kosaraju's`. The x-axis also shows the amount of n and m, and y-axis also shows the average running times. It can be seen that all three solvers solve the *Black-and-White 2-SAT* problems in linear time.

In the followings we show some comparisons on sparse and dense graphs. We investigated that if the graph is not strongly connected, the running time depends on our luck in finding the first branch. These results are presented in Table 4.6 and Table 4.7. The first column presents the node size of network-based instances. Note, that we randomly generated 100

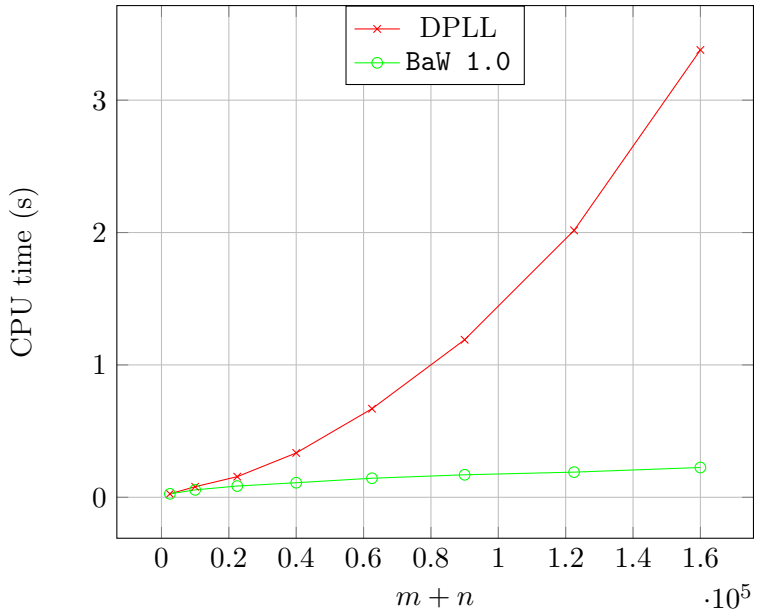


Figure 4.1: Compared to the direct implementation of `DPLL` and the `BaW 1.0`

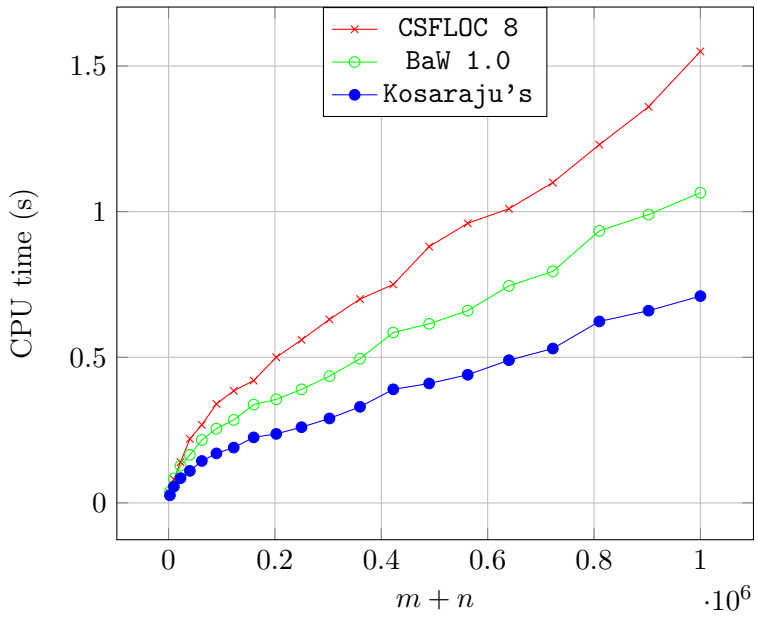


Figure 4.2: Compared to the `CSFLOC 8`, `BaW 1.0` and the `Kosarju's` algorithm

instances per all node size (2000,5000,10000). In the second column shows the maximum amount of the number of nodes *(vertices)* and clauses *(edges)*, if the graph is a complete graph. The next two columns give information about average densities and reference values. Finally, the last column shows the achieved speed decrease with *DFB*. Two additional notes to the tables: first, the reference values are 1 everywhere; second, in the case of *DFB*, the

searches were started randomly from the largest `source` or `sink`. The last column shows these deviations from this reference values. We found that in the case of not strongly connected and sparse graphs, an average of 10% speedup could be achieved, despite the increased preprocessing time.

Inst. (n)	max n+m	avg. dens.	basic	with DFB
2000	4e6	5-10 %	1	92,6%
5000	2.5e7	5-10 %	1	88,1%
10000	1e8	5-10 %	1	86,7%

Table 4.6: All instances are *SAT (not strong connected graphs)*

Inst. (n)	max n+m	avg. dens.	basic	with DFB
2000	4e6	11-20 %	1	94,2%
5000	2.5e7	11-20 %	1	90,7%
10000	1e8	11-20 %	1	87,1%

Table 4.7: All instances are *SAT (not strongly connected graphs)*

4.4 *SAT* Solving on Grids

In this section, we present our parallel *SAT* solver `CCGrid` *(Cube and Conquer on Grid)* on computational grid infrastructure. Our goal is to develop techniques for using distributed computing resources to efficiently solve instances of the propositional satisfiability problem (*SAT*). We claim that computational grids provide a distributed computing environment suitable for *SAT* solving. We apply the Cube and Conquer approach to *SAT* solving on grids. Our solver consists of two major components. The master application runs `march_cc`, which applies a lookahead *SAT* solver in order to partition the input *SAT* instance into work units distributed on the grid. The client application executes an `iLingeling` instance, which is a multi-threaded *CDCL SAT* solver. We use `BOINC` middleware, which is part of the `SZTAKI Desktop Grid` package and supports the Distributed Computing Application Programming Interface (*DC-API*).

4.4.1 Architecture

Our application is a variant of the Cube and Conquer approach [72] and consists of two major components: a master application and a client application. The master is responsible for dividing the global input data into smaller chunks and distributing these chunks in the form of work units. Interpreting the output generated by the clients out of the work units and combining them to form a global output are also the job of the master. The

architecture is depicted in Figure 4.3. Similar to [147], the environment for running our system is the SSZTAKI Desktop Grid [90] and BOINC [10], and was implemented by the use of the *DC-API*.

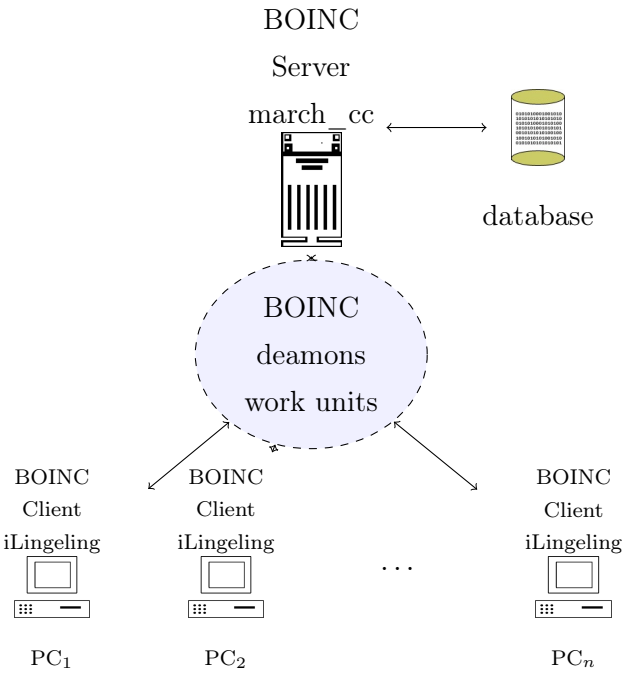


Figure 4.3: CCGrid architecture

4.4.2 The Master

The master executes a partitioning tool called `march_cc` [72], which is based on the lookahead *SAT* solver `march`. Given a *CNF* file, `march_cc` primarily tries to refute the input clause set. If this does not succeed, `march_cc` outputs a set of assumptions *(cubes)* that describe the cutoff branches in the *DPLL* tree. These assumptions cover all subproblems of the input clause set that have not been refuted during the partitioning procedure. Given these assumptions, the master application creates work units, each of which consists of the input *CNF* file and a slice of the assumption set. If one of the clients reports one of the work units to be satisfiable, then the master outputs the satisfying model and destroys all the running work units. If every client reports unsatisfiability, then the master outputs unsatisfiability.

The pseudocode below shows how the master application works. It is divided into three procedures; the Main procedure is shown in Algorithm 6. It shares two constants with the other procedures: (i) *maxAsmCount* defines the maximum number of assumptions per work unit; (ii) *rfsInterval* gives a refresh interval at which *DC-API* events are processed. The master application uses several global variables; all of them are self-explanatory. In loop 6-9, work units are created, by calling the procedure CreateWorkUnit. Loop 10-13

then processes *DC-API* events generated by those work units that have finished solving their subproblems. Processing *DC-API* events is done by calling a callback function which has been previously set to ProcessWorkUnitResult in line 3. The loop stops if either one of the work units returns a *SAT* result or all the work units completed.

Algorithm 6 Master: main procedure

Require: global constants *maxAsmCount*, *rfsInterval*
Require: global variables *cnfFile*, *asmFile*, *wuCount*, *res*, *resFile*

```
 1: procedure Main
 2:     initialize DC-API master
 3:     set ProcessWorkUnitResult as result callback
 4:     open asmFile
 5:     wuCount ← 0
 6:     while not EOF(asmFile) do
 7:         CreateWorkUnit
 8:         wuCount ← wuCount + 1
 9:     end while
10:     while res ≠ SAT and wuCount > 0 do
11:         wait rfsInterval
12:         process DC-API events
13:     end while
14:     if res ≠ SAT then
15:         res ← UNSAT
16:         cancel all work units
17:     end if
18: end procedure
```

CreateWorkUnit, shown in Algorithm 7, creates and submits a work unit to the grid. First, the *CNF* file is added to the new work unit. Then, in the loop, at most *maxAsmCount* assumptions from *asmFile* are copied into the new file *asmChunkFile*. Note that *asmFile* is global, it has been opened by the Main procedure (Algorithm 6, line 4), and therefore its current file position is held. Finally, *asmChunkFile* is added to the work unit, which is then submitted to the grid.

As already mentioned, ProcessWorkUnitResult, shown in Algorithm 8, works as a callback function for *DC-API* events. It processes the result returned by a work unit.

The Client

Each client executes the parallel *CDCL* solver `iLingeling` [72, 24], for a fixed number of threads. Each thread executes a separate `lingeling` instance. `iLingeling` expects as input an *iCNF* file, including 1 or more assumptions, which is then loaded into a working queue. Each `lingeling` instance reads the input clause set, and then, in each iteration, gets the first assumption from the working queue.

If one of the `lingeling` instances can prove that the clause set is satisfiable under the

Algorithm 7 Master: creating work units

```
 1: procedure CreateWorkUnit
 2:     wu ← new work unit
 3:     wu.cnfFile ← cnfFile
 4:     asmChunkFile ← new file
 5:     for i ← 1 to maxAsmCount do
 6:         if EOF(asmFile) then
 7:             break
 8:         end if
 9:         copy next assumption from asmFile to asmChunkFile
10:         i ← i + 1
11:     end for
12:     wu.asmFile ← asmChunkFile
13:     submit wu to the grid
14: end procedure
```

Algorithm 8 Master: processing work unit result

```
1: procedure ProcessWorkUnitResult(wu)
2:     if wu.res = SAT then
3:         res ← SAT
4:         copy wu.resFile to resFile
5:     end if
6:     wuCount ← wuCount − 1
7: end procedure
```

given assumptions, then `iLingeling` reports that the clause set itself is satisfiable, the satisfying model is returned, and hence the remaining assumptions in the working queue can be ignored. Otherwise, i.e., if a `lingeling` instance reports unsatisfiability, then the assumption is retrieved from the working queue and the same *SAT* solver instance continues with the solving procedure. If the working queue becomes empty, then `iLingeling` reports that the clause set under the given set of assumptions is unsatisfiable.

Algorithm 9 shows the client's MAIN procedure. It uses one global constant, *thrCount*, which specifies the number of worker threads to use. First, the procedure creates an `iLingeling` instance with *thrCount* worker threads, loads both the *CNF* and the assumption files, and runs `iLingeling`. In loop 7-12, the results by all the threads are checked: if any of them is *SAT* then the result for the work unit is *SAT*; otherwise it is UN*SAT* (line 14). The result, as well as the satisfying model, is written into a result file by the procedure CreateResultFile, shown in Algorithm 10.

4.4.3 Results and Testing Environment

Our implementation consists of a quad-core `SUN` server with 6 GB memory, used as a master, and 20 quad-core *PCs* with 2 GB memory, used as clients. In our experiments,

Algorithm 9 Client: main procedure

Require: global constant *thrCount*

```
1: procedure MAIN(wu)
2:     initialize DC-API client
3:     iLingeling ← new iLingeling instance using thrCount threads
4:     load wu.cnfFile into iLingeling
5:     load wu.asmFile into iLingeling
6:     run iLingeling
7:     for i ← 1 to thrCount do
8:         if ith thread's result is SAT then
9:             wu.res ← SAT
10:            break
11:        end if
12:    end for
13:    if i > thrCount then
14:        wu.res ← UNSAT
15:    end if
16:    CREATERESULTFILE(wu, iLingeling)
17: end procedure
```

Algorithm 10 Client: creating result file

```
1: procedure CREATERESULTFILE(wu, iLingeling)
2:     resFile ← new file
3:     write wu.res into resFile
4:     if wu.res = SAT then
5:         model ← satisfying assignment from iLingeling
6:         write model into wu.resFile
7:     end if
8:     wu.resFile ← resFile
9: end procedure
```

we used instances from the *SAT* Challenge 2012, from the Application (*SAT* + UN*SAT*) and the Hard Combinatorial (*SAT* + UN*SAT*) tracks. Results are presented in Table 4.8 and Table 4.9. The 1st column represents the instance's name. In the 2nd column, **A** resp. **HC** denotes Application resp. Hard Combinatorial problems. The 4th column shows the number of cubes, generated by `march_cc`. The 3rd resp. the 5th column shows the runtime of `march_cc` resp. `iLingeling`, being executed on the master. The 6th column contains the sum of the previous two numbers, which represents the overall runtime of the cube-and-conquer approach running on a single *(quad-core)* machine. The total runtime of `CCGrid` is shown in the 7th column, while the 8th column measures the speedup as the ratio of the runtimes in the 6th and 7th columns.

In our approach, `CCgrid` has been executed without any communication among clients. Even though they do not cooperate and do not exchange learnt clauses, `CCGrid` shows

		march_cc	# of cubes	iLingeling	cube-and-conquer	CCGrid	speedup
# of clients		**1**		**1**	**1**	**20**	
vmpc_26	**A**	8.38	296	40.22	48.6	13.64	**3.56**
AProVE09-07	**A**	65.93	4245	19.12	85.05	79.16	**1.07**
clauses-4	**A**	29.68	25	59.01	88.69	81.98	**1.08**
gss-16-s100	**A**	155.28	6292	201.21	356.49	171.71	**2.08**
IBM_FV_2004_rule_batch_22_*SAT*_dat.k65	**A**	17.93	361	148.95	166.88	154.02	**1.08**
ezfact64_3.sat05-450.reshuffled-07	**HC**	458.71	469428	63.71	522.42	505.72	**1.03**
sgen1-sat-160-100	**HC**	10.65	210168	419.92	430.57	62.28	**6.91**
em_7_4_8_exp	**HC**	20.06	19419	170.9	190.96	46.47	**4.11**
battleship-16-31-sat	**HC**	174.89	91757	2.69	177.58	180.89	**0.98**
Hidoku_enu_6	**HC**	125.02	256225	91.61	216.63	159.31	**1.36**

Table 4.8: Runtimes and speedup
all instances are *SAT*

a wide range of speedups. We achieved speedup up to ca. 8.5 on UN*SAT* instances (*QG-gensys-icl003.sat05-2715.reshuffled-07*) and up to ca. 7 on *SAT* instances (*sgen1-sat-160-100*).

Since the master has to distribute quite large work units over the network, communication overhead matters in the case of small instances, where communication costs are significant compared to the input size. Therefore, although we used a 1Gbps LAN in our experiments, cube-and-conquer running on a single machine outperformed `CCgrid` on some instances. If we look at the *battleship-16-31-sat* row in Table 4.8, we can see that `march_cc` and `iLingeling` can solve this problem on 1 client a bit faster than `CCGrid` on 20 clients.

In the case of satisfiable instances, we might be lucky, finding a model quickly or unlucky. If there are many satisfying models, then it is not worth to distribute the problem over many clients. However, if there exist only a few models, then it is a good idea to use many clients, since the more clients we use, the more probable it is for a client to be lucky enough to find one of those few solutions. Unfortunately, we have no information about how many models the instances in Table 4.8 have.

`CCGrid` seems to be much better in distributing satisfiable instances from the **HC** track than the ones from the **A** track, since `march_cc` seems to generate much more cubes for the previous ones.

		march_cc	# of cubes	iLingeling	cube-and-conquer	CCGrid	speedup
# of clients		**1**	**1**	**1**	**20**		
counting-clqcolor-unsat-set-b-clqcolor-08-06-07.sat05-1257.reshuffled-07	**A**	5.77	112757	12.77	18.54	8.71	**2.13**
gensys-ukn002.sat05-2744. reshuffled-07	**A**	12.70	21408	230.01	242.71	71.55	**3.39**
Q32inK09	**A**	12.15	5279	35.89	48.04	14.38	**3.34**
QG6-dead-dnd002.sat05-2713. reshuffled-07	**A**	2.38	12147	35.79	38.17	4.74	**8.05**
QG-gensys-icl003.sat05-2715.reshuffled-07	**A**	25.43	38466	291.05	316.48	37.24	**8.49**
instance_n6 i7_pp_ci_ce	**A**	103.78	29290	111.72	215.50	117.20	**1.84**
AProVE07-09.cnf	**A**	34.43	86048	4.55	37.98	37.46	**1.01**
battleship-10-10-unsat	**HC**	0.36	2317	15.47	15.83	4.48	**3.53**
rand_net60-40-10.shuffled	**HC**	111.23	130227	13.24	124.47	115.62	**1.08**
smtlib-qfbv-aigs-ext_con_032 008_0256-tseitin	**HC**	67.24	22384	7.29	74.53	71.47	**1.04**

Table 4.9: Runtimes and speedup
all instances are *UNSAT*

In the case of unsatisfiable instances, we cannot be lucky to find an early solution since there is no satisfying model. When comparing the speedups in Table 4.8 and Table 4.9, we can see that speedups around 1 are more frequent on satisfiable instances.

This shows that in the case of unsatisfiable instances there is less risk of wasting resources without any speedup.

4.5 Conclusions

In this chapter first, we have introduced the `CSFLOC` and the `BaW 1.0` *SAT* solvers. We also have presented a parallel *SAT* solver `CCGrid`, which runs on the MTA `SZTAKI Grid` using `BOINC`. At the beginning of the section, we have introduced the `CSFLOC` algorithm which counts full-length clauses.

After that, we have introduced a problem-specific *SAT* (`BaW 1.0`) solver that is two times faster than `CSFLOC` for *Black-and-White 2-SAT* problems. We have described the main components of `BaW 1.0`. We have shown with the modified unit propagation how the *Black-and-White 2-SAT* problems in linear time can be checked. After that, we have presented our experiments on these network-based communication benchmarks. We have shown that in the case of not strongly connected and sparse graphs, an average of 10% speedup can be achieved. We intend to investigate how to further analyze the communication graph using `BaW 1.0`. A further goal is to extract the `BaW 1.0` to find 2-Edge and 2-Vertex Strongly Connected Components in Quadratic Time.

At the end of this chapter, we presented the parallel *SAT* solver `CCGrid`, which runs on the MTA `SZTAKI Grid` using *BOINC*. In this version, the master application applies `march_cc`, using a lookhead solver, to split a *SAT* instance. The client application uses the parallel *SAT* solver `iLingeling` to deal with several assumptions. The client creates a separate `iLingeling` instance for each work unit, and destroys it after completing the work unit. For the sake of improving our current results, in future work, we would like to preserve the state of `iLingeling` instances, including learnt clauses. In our experience, the cube generation phase implemented in `march_cc` makes up a significant part of the runtime. As a consequence, we were mostly able to achieve significant speedup on such instances on which the cube generation phase took a relatively short time. In order to achieve larger speedup on unsatisfiable instances, it might be useful to call `march_cc` not only while partitioning the original problem, but also for repartitioning difficult subproblems, e.g., those on which a client exceeds a certain time limit.

5 Part III

First, in this chapter, we introduce an *SMT* formalization for *WSN*s that provides a single-hop *WSN* simulation environment with one of the most common wireless sensor node types. By applying an *SMT* formalization for *WSN*s, a sleep/wake-up scheduling that respects several reliability and security requirements can be generated by using *SMT* solvers. The recent success of *OMT* approaches makes us able to generate for a *WSN* a sleep/wake-up scheduling that is optimal in terms of energy efficiency. We show how to use it for *SMT* formalization to optimize network lifetime. We propose several benchmarks and we present our experiments with the most relevance *OMT* solvers: `Z3`, `OptiMathSAT` and `Symba`. In order to better understand the problem, we have examined the correlation between the certain graph criteria and the degree of challenge for *OMT* solvers. We show how criteria helps in a real-world environment to plan the physical distribution of sensor nodes in order to prolong the *WSN*'s lifetime as much as possible. Hereupon, we show an idea of speeding up the *OMT* solving process by taking into consideration some resources in the systems and by applying regression analysis on those resource values. Finally, we present a novel *OMT* solver called `Puli`.

5.1 Introduction

WSNs consist of a number of spatially distributed sensor nodes, which cooperatively monitor physical or environmental conditions. For testing, we provide a real single-hop *WSN* simulation environment and relevant testing verdicts through our tool. An average sensor node consists of a microcontroller and a *RF* transceiver. Micro controller performs computation/organization and administration. A radio transceiver is used to send and receive data packets whose contents can vary from sensor information, network topology updates, to over-the-air firmware updates. We choose one of the most common wireless sensor node type commonly known as mote. Mote is used with *PCB* antenna like `MICAz`. This type is easy to implement and can be used to build *WSN* for various applications.

As already mentioned earlier *(see in Chapter 2.2)*, in order to ensure the dependability of *WSN* functionalities, several reliability and security requirements have to be fulfilled. In this chapter, we primarily focus on military applications. These applications have some special features, one of these that the deployment may require a sensor node to avoid monitoring the same target point for a long time, otherwise the sensor node can easily be damaged, detect or attack. Some additional area-specific requirements: *partial*

coverage [113], the *evasive constraint* [18], and the *moving target constraint* [152].

Most of the previous studies on lifetime maximization for *WSN*s model the problem as an optimization problem and solve it using different heuristic algorithms [176, 38, 40]. The size of the tested problems is generally scale up to a few hundreds of sensor nodes [176] and a few tens of target points [38], which sometimes comes with the price of losing 100% precise coverage. The *SMT*-based representations and the *SMT* solvers provide an effective tool to solve constraint satisfaction problems in diverse application areas including software and hardware verification, planning, scheduling, etc. A few previous works apply *SMT* formalization to the aforementioned *WSN* constraints and use *SMT* solvers to generate an appropriate sleep/wake-up scheduling for a *WSN*. There are no complex solutions on this research-area yet, for example Weiqiang et. al. [97] focuses only on the coverage problem. Besides the coverage problem, the partial coverage, the evasive constraint, and the moving target constraint [152] are also explored. For the experiments in [152], the *SMT* solver `Yices` is utilized, and the results show that the approach can scale up to hundreds of sensor nodes. Note that both [97, 152] investigate generating a sleep/wake-up scheduling that fulfills the aforementioned constraints, but none of them addresses the maximization problem of the *WSN* lifetime. Thanks to the development of the *OMT*-based solutions it makes us able to generate a sleep/wake-up scheduling that provides maximal lifetime for the *WSN*, while keeping all the flexibility and strength of the *SMT*-based approaches, namely that different dependability and security constraints can be combined on demand. This makes this kind of *WSN* optimization an excellent application for *OMT* solvers. To the best of our knowledge, no scientific work has ever reported any usage of *OMT* solvers for *WSN* optimization.

5.2 *OMT*-based Representation

5.2.1 Notations

Each sensor node has a predefined sensing range which can be heterogeneous in our model. For later understanding, we introduce the following notations:

- n: the number of sensor nodes,
- m: the number of target points,
- r_i: the sensing range of the i^{th} node,
- L_i: the lifetime of the i^{th} node,
- $d_{i,j}$: the distance between the i^{th} node and the j^{th} point,
- T : the lifetime of the *WSN*,
- $w_{i,t}$: Boolean variable that denotes if the i^{th} node is awake at the t^{th} time interval.

5.2.2 Test Environment and Technical Background

For simulating *WSNs*, we chose an `IEEE 802.15.4` compatible sensor node that is able to communicate wirelessly and has common parameters such as a $3V$ power supply [1]. Such sensor nodes are provided with an RF transceiver with an estimated range of $100 - 120m$.

To accurately define the different ranges for different performance levels, we chose a commonly used RF transceiver, the `CC2420` [2].

Technical data (`CC2420`):

- Sleep mode: $300\mu A, 1MHz, 2V$
- Standby mode: $0.1\mu A$
- Transmission speed: $250Kbps$
- RF power: $-25dBm$ - $0dBm$
- Range: $120m$ outdoor, $20 - 30m$ indoor
- Power consumption: Rx: $18.8mA$, Tx: $17.4mA$, Sleep mode: $1\mu A$

Although, depending on the operation system of the sensor node, 32 to 256 different performance levels can be defined, the manufacturer of `2420` published data only about 8 performance levels. For sensor node simulation, we need to know the power consumption and the sensing range for each performance level.

In Table 5.1, data about those performance levels are given, where the estimated ranges can be calculated as follows.

PA_LEVEL	Output Power (dBm)	Output Power (mW)	Current Consumption (mA)
31	0	1,000	17,4
27	-1	0,794	16,5
23	-3	0,501	15,2
19	-5	0,316	13,9
15	-7	0,200	12,5
11	-10	0,100	11,2
7	-15	0,032	9,9
3	-25	0,003	8,5

Table 5.1: Estimated ranges for the different performance levels of a sensor node equipped with a `CC2420` RF transceiver.

Euclidean distance between transmitter i (x_i,y_i) and receiver *(target)* j (x_j,y_j):

$$d_{(i,j)} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}. \tag{5.2.1}$$

[1] `http://www.memsic.com/userfiles/files/Datasheets/\textit{WSN}/micaz_datasheet-t.pdf`.
[2] `http://www.ti.com/lit/ds/symlink/cc2420.pdf`

Maximum range ($d_{max} = 120m$) for the maximum output power ($P_{max} = 1mW$) of the transmitter.
The output power $P(d)$ is directly proportional with the square of range d:

$$P(d) = pd^2 \tag{5.2.2}$$

where p is the power density. We can calculate the value of p from the maximum range $d_{max} = 120m$ and the maximum output power $P_{max} = 1mW$ as follows:

$$p = \frac{P_{max}}{d_{max}^2} = \frac{1}{120^2} \approx 6.94 \cdot 10^{-5} mW/m^2 \tag{5.2.3}$$

From this and the minimum output power $P_{min} = 3.16\mu W$, one can easily calculate the minimum range:

$$d_{min} = \sqrt{\frac{P_{amp}}{p}} = \sqrt{\frac{3{,}16 \cdot 10^{-6}}{6{,}94 \cdot 10^{-5}}} = 6{,}75m. \tag{5.2.4}$$

The estimated range can be calculated according to the distances (d_{min},d_{max}) and the power consumption (I_{min}, I_{actlev}) of the power levels (P_{min},P_{max}) as follows:

$$d_{est} = \frac{d_{max} - d_{min}}{P_{max} - P_{min}}(I_{actlev} - I_{min}) + d_{min}. \tag{5.2.5}$$

5.2.3 *SMT* Formalization

For representing the lifetime of sensor nodes and the time intervals, the best choice is obviously to apply the theory of integer and real numbers. Similarly, it is sufficient to represent the sensor ranges and the distances between sensors nodes and points as integer numbers. The avoiding the use of non-linear arithmetic makes the solving process more feasible, so does the avoiding of quantifiers. All the variables whose value depends on T are represented as uninterpreted functions [3].

We define the following constraints on a *WSN*:

- **Lifetime constraint.** For each sensor node, the number of time intervals at which the node is awake must not exceed the node's lifetime.

$$\forall i \ (1 \leq i \leq n). \ \sum_{t=1}^{T} w_{i,t} \leq L_i \tag{5.2.6}$$

- **Coverage constraint [176, 38, 40].** Every point must be covered by at least $K \geq 1$ sensor nodes at any time.

$$\forall j,t \ (1 \leq j \leq m, \ 1 \leq t \leq T). \ \sum_{i \in S_j} w_{i,t} \geq K \tag{5.2.7}$$

where

$$S_j = \{i \mid d_{i,j} \leq r_i\}. \tag{5.2.8}$$

[3] Uninterpreted functions can be eliminated by introducing Ackermann constraints, which comes with the price of quadratic growth in problem size. Consequently, we have decided to apply the logic `qfuflia`

- **Evasive constraint [18, 152].** Each sensor node must stay active for more than $E \geq 1$ consecutive time intervals.

$$\forall i,t\ (1 \leq i \leq n,\ 1 \leq t \leq T - E).\ \sum_{t'=t}^{t+E} w_{i,t'} \leq E \tag{5.2.9}$$

- **Moving target constraint [152].** Some of the points are considered to be critical points, which must not to be covered by the same sensor for more than $M \geq 1$ consecutive time intervals.

$$\forall j \in \{\leq j \leq m\},\ \forall i \in S_j,\ \forall t\ (1 \leq t \leq T - M).\ \sum_{t'=t}^{t+M} w_{i,t'} \leq M \tag{5.2.10}$$

Note that all the aforementioned constraints apply linear arithmetic. All the variables whose value depends on T are represented as uninterpreted functions. For the sake of the feasibility of the solving process, it is advisable to avoid the use of quantifiers by unrolling all the constraints. Therefore, all the aforementioned constraints can be encoded as `qfuflra` formulas.

The lifetime constraint can be formalized in *SMT-LIB* as follows:

```
( <=
(+ ( boolToInt ( w i 0))
( boolToInt ( w i 1))
     ( boolToInt ( w i T )) )
( L i ) )
```

The *K-coverage* constraint can be formalized in *SMT-LIB* as follows, for all target points j and time interval t:

```
(>=
  (+  (boolToInt (covers0jAt t)) (boolToInt (covers1jAt t))
  ... (boolToInt (coversnjAt t)) )
  K )
```

where the function `(coversijAt t)` tells if the sensor node i covers the target point j at the time interval t. A node must not stay awake for more than $E \geq 1$ consecutive time intervals. Finally, we show how to formalize the *evasive constraint* with parameter E. The following assertion should be added for all sensor nodes i and time intervals $t \leq T - E$:

```
(<=
  (+  (boolToInt (w i t)) (boolToInt (w i t+1))
  ... (boolToInt (w i t+E)) )
  E )
```

The moving target constraint can be formalized in a similar manner.

5.2.4 *SMT* Extractions for *OMT* Solvers

Since the *SMT-LIB* formalization of the corresponding optimization problem slightly differs for the different *OMT* solvers, we generated three variants of each benchmark instance: one for the `OptiMathSAT`, one for the `z3`, and one for the `Symba`. Our main goal is to maximize the *WNS*'s lifetime.
The different *SMT* extractions to the *OMT* solvers are the following:

- for the `Z3` the formalization is extremely simple:

  ```
  ( maximize T )
  ```

- for the `OptiMathSAT` the lower bound and the upper bound must be specified, too, as follows:

  `( maximize T : local - lb 0 : local - ub` $\hat{T}$`)`

  where $\hat{T} = \sum_{n}^{i=1} L_i$ works as a time horizon for the *WSN*,

- for the `Symba` requires a completely different formalization [4], as follows:

  ```
  (= >  $ constraints ( <= T TOpt ) )
  ```

  where `TOpt`, similar to `T`, is a variable declared by us.

5.2.5 *OMT* Benchmarks

In our simulations, the performance level for each sensor node was set randomly in advance, and then the corresponding sensing range was used throughout the simulation.

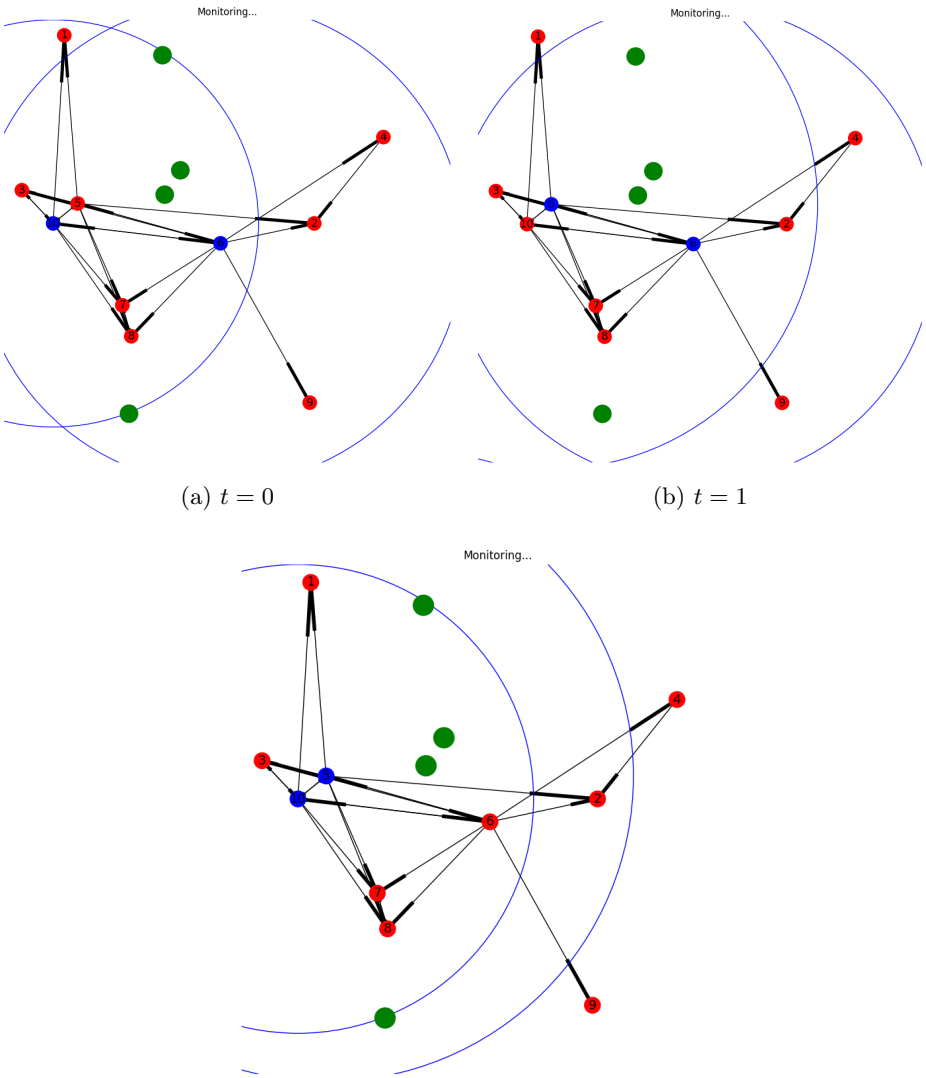
In the *WSN* simulation, we generated a network grid of 600 meters by 600 meters. The physical locations of the sensor nodes and the target points were set randomly, as well as the performance levels of the sensor nodes. We generated two benchmark sets from the simulator:

- Harder benchmarks with 10 sensor nodes, 4 target points, 2-coverage constraint, evasive constraint with $E = 3$, and moving target constraint with $M = 2$.
- Easier benchmarks with 10 sensor nodes, 2 target points, 1-coverage constraint, evasive constraint with $E = 2$, and moving target constraint with $M = 1$.

Within each benchmark set, we generated

- 20 benchmarks with all the constraints enabled,
- 20 benchmarks with only the moving target constraint disabled, and
- 20 benchmarks with only the evasive constraint disabled.

[4] The objective should be written as an implication with all the constraints of interest on the left-hand side and the following inequality on the right-hand side.



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$

Figure 5.1: Sleep/wake-up scheduling of sensor nodes for 2-coverage and evasive constraint with $E = 2$. The active nodes *(blue dots)* are monitoring the target points *(green dots)*

.

An Example

Figure 5.1 illustrates how sleep/wake-up scheduling works with 2-coverage. The blue dots represent the active sensor nodes, the red dots the ones in sleep mode and the green dots the target points. Blue circles show the sensing ranges of the active sensors. Note that in each time interval, each target point is monitored by *(at least)* 2 sensor nodes.

In some applications, sensors are deployed in a hostile environment or in critical systems [152], thus it might be important to protect the sensors from being active for too long.

5.2.6 Experiments and Results

In this subsection, we describe the implementation details used to simulate *WSNs* and to generate benchmarks with different sets of constraints with different parameter values. The simulations and experiments were run on 3.60 GHz 8-core *CPU* with 8 *GB* memory. The wall clock time limit was set to 600 seconds and the memory limit to 3 *GB*.

Since the strength of our *OMT*-based approach is that any constraints can be combined with the *WSN* optimization problem. It is not involved in our experiments classical *WSN* optimization approaches [176, 38, 40] that can only deal with the coverage constraint. Our main goal is to examine the impact of the different dependability and security constraints to the *WSN* lifetime maximization problem, as well as the impact of critical parameters such as the number of sensor nodes, the number of target points, the parameters for the coverage, for the evasion and for the moving target constraints, respectively.

Table 5.2 provides an overview of the evaluation results for the harder benchmarks. The total number of solved *SAT/UNSAT* instances is shown, as well as the number of timeouts (*#TO*), the highest maximal lifetime (*Opt*) that was found for the given benchmark, the average runtime *(CPU time-sec.)*, the average memory consumption *(MB)* and the number of crashes *(where relevant)*. Although *Opt* is not a representative value, we use it to illustrate the magnitude of the *WSN* optimization problem that the given solver is able to cope with.

	Solver	*#SAT/UNSAT*	*#TO*	*Opt*	*Time*	*Space*	*#Crash*
All constraints on	`OptiMathSAT`	11/5	4	73	245.5	440.3	
	`Z3`	2/5	13	72	393.6	449.8	
	`Symba`	1/5	14	2	423.1	461.9	
Moving target off	`OptiMathSAT`	10/5	5	74	215.8	314.8	
	`Z3`	7/5	8	73	258.2	408.3	
	`Symba`	4/5	11	60	393.5	439.1	
Evasive off	`OptiMathSAT`	12/4	3	74	149.0	286.5	1
	`Z3`	7/5	8	72	265.2	468.5	
	`Symba`	5/5	10	60	355.6	475.0	

Table 5.2: Results for qfuflia benchmarks with 10 sensor nodes, 4 target points, 2-coverage, an evasive constraint with $E = 3$ and a moving target constraint with $M = 2$.

The Table 5.2 clearly shows that the `OptiMathSAT` outperforms the other two solver on the harder benchmarks, although it crashes on one instance with a segmentation fault. It is very interesting that the `OptiMathSAT` does not seem to be sensitive to what type of constraints are involved in the *WSN* optimization problem, regarding the number of solved instances. On the other hand, runtime and memory consumption decrease if one of the constraints is disabled, especially in the case of the evasive constraint.

When all the constraint are enabled, Z3 and Symba can solve significantly fewer instances than the OptiMathSAT does. When one of the constraints is disabled, the performance of Symba gets slightly better, although their runtime and memory consumption do not really change. Let us note that we experienced insignificant runtimes on all the *UNSAT* instances by all the solvers, this is why they can solve all the *UNSAT* instances, except for the case of crash.

Table 5.3 shows the evaluation results for the easier benchmarks, which consist of *SAT* instances only. Here, the Z3 is clearly the winner since it can solve all the instances at very moderate runtime. Note that the greatest maximal lifetime found by the solvers is 226, therefore we find Z3's performance on those benchmarks quite remarkable.

	Solver	*#SAT/UNSAT*	*#TO*	*Opt*	*Time*	*Space*
All constraints on	Z3	20/0	0	226	63.1	493.1
	OptiMathSAT	12/0	8	159	277.8	520.7
	Symba	9/0	11	159	484.9	436.2
Moving target off	Z3	20/0	0	226	49.2	333.7
	OptiMathSAT	11/0	9	130	311.3	476.1
	Symba	9/0	11	159	488.8	340.0
Evasive off	Z3	20/0	0	226	50.8	300.0
	Symba	19/0	1	226	324.8	334.1
	OptiMathSAT	12/0	8	159	344.1	488.2

Table 5.3: Results for qfuflia benchmarks with 10 sensor nodes, 2 target points, 1-coverage, an evasive constraint with $E = 2$ and a moving target constraint with $M = 1$.

Interestingly, the OptiMathSAT does not provide better results than the ones on the harder benchmarks. On the contrary, average runtime and memory consumption have even increased. Surprisingly, the Symba remarkably accelerates with the evasive constraint disabled. The Symba not only outperforms the OptiMathSAT, but it almost reaches the performance of Z3 in terms of solved instances, although Symba's average runtime is still much greater than the one of Z3.

It is also an interesting question if the theory of integer numbers is the best choice. Therefore, we decided to rerun the experiments for the easier benchmarks over real numbers (qfuflra). The results in Table 5.4 show that the Z3 and the OptiMathSAT provide almost the same performance as before, while the Symba accelerates drastically and can solve almost all the instances.

Summary To summarize our experiments, the OptiMathSAT provides the most stable performance and scales the best for the presence/absence of different constraints and for different parameter settings. Using Z3 is advantageous on instances with fewer target

points and lower parameter values, for which it provides amazing runtimes. Although the `Symba` scales the worst over integer numbers, it provides convincing performance over real numbers.

	Solver	*#SAT/UNSAT*	*#TO*	*Opt*	*Time*	*Space*
All constraints on	`Z3`	19/0	1	226	87.7	497.1
	`Symba`	18/0	2	226	223.4	578.5
	`OptiMathSAT`	12/0	8	159	277.0	507.4
Moving target off	`Z3`	20/0	0	226	41.4	318.9
	`Symba`	20/0	0	226	172.5	393.0
	`OptiMathSAT`	11/0	9	130	310.6	469.9
Evasive off	`Z3`	20/0	0	226	36.2	284.4
	`Symba`	20/0	0	226	128.9	340.5
	`OptiMathSAT`	11/0	9	159	345.2	467.5

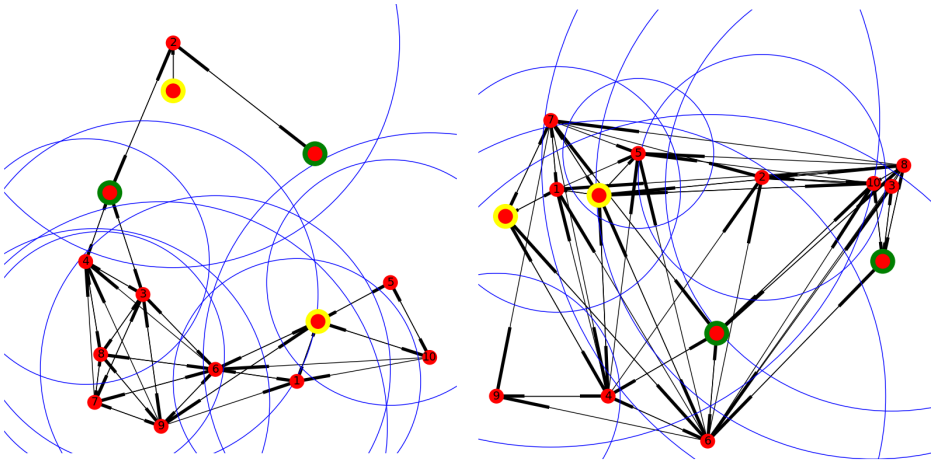
Table 5.4: Results for `qfuflra` benchmarks with 10 sensor nodes, 2 target points, 1-coverage, an evasive constraint with $E = 2$ and a moving target constraint with $M = 1$.

5.3 Proposing Certain Criteria for a Graph

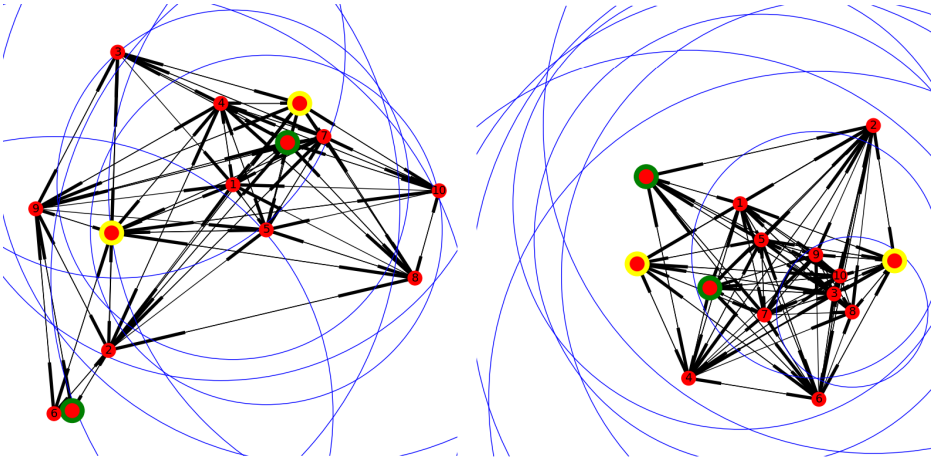
In this section, we address this problem by proposing certain criteria for a graph of sensor nodes and target points. We show that there is correlation between the degree of fulfillment of those graph criteria and the degree of challenge for *OMT* solvers. We also discuss why the examination of those graph criteria helps in a real-world environment to plan the physical distribution of sensor nodes in order to prolong the *WSN's* lifetime as much as possible.

5.3.1 Graph Properties for Wireless Sensor Networks

Let $\mathcal{S} = [1,n]$ be the set of sensor nodes in a *WSN*. By a *communication graph* we mean a directed graph $(\mathcal{S};\mathcal{E})$ where $\mathcal{E} \subseteq \mathcal{S} \times \mathcal{S}$ (see in Chapter 3.3). In graph theory, the density of a graph $(\mathcal{V};\mathcal{E})$ can be calculated as $\frac{|\mathcal{E}|}{|\mathcal{V}|(|\mathcal{V}|-1)}$ [48]. Since the number of edges for a complete directed graph is $|\mathcal{V}|\,(|\mathcal{V}|-1)$, the maximum density is 1. Clearly, the minimum density is 0 (for empty graphs). For a *WSN*, it is a natural requirement to be *connected*. Therefore, a minimally dense communication graph is actually a tree, for which $|\mathcal{E}| = |\mathcal{S}| - 1$. Consequently, the minimal density is $\frac{1}{|\mathcal{S}|}$. For instance, if the *WSN* consists of 10 sensor nodes, the minimum density is 10%. We define two sparse (20%-30%, 40%-50%) and two dense groups (60%-70%, 80%-90%) for a test on *WSN*s. Figure 5.2 tries to



(a) $20 - 30\%$ density (b) $40 - 50\%$ density.



(c) $60 - 70\%$ density. (d) $80 - 90\%$ density.

Figure 5.2: *WSN*s of different graph density.

visually capture the difference between those cases, where dots represent the sensor nodes and blue circles the sensing ranges.

Based on preliminary tests, we found that the lifetime of a *WSN* depends not primarily on the density of the communication graph, but rather on the number of nodes that observe a target. Therefore, we extend the communication graph with the target points $\mathcal{T}$. Since a sensor node observing a target point can be considered as a kind of two-way communication between the sensor and the target, the *extended communication graph* basically merges the original directed graph with an undirected one, as follows.

Definition 1 (Extended Communication Graph)**.** *Given a communication graph* $(\mathcal{S};\mathcal{E})$ *and a set of target points* $\mathcal{T}$*, the extended communication graph is defined as* $(\mathcal{S}';\mathcal{E}')$ *where*

$$\begin{aligned} \mathcal{S}' &= \mathcal{S} \cup \mathcal{T} \\ \mathcal{E}' &= \mathcal{E} \cup \left\{(i,j),\ (j,i) \;\middle|\; i \in \mathcal{S},\ j \in \mathcal{T},\ d_{i,j} \leq r_i\right\} \end{aligned}$$

In Figure 5.2, large *(green and yellow)* dots represent the target points. By looking at

the figure, it seems that the denser an extended communication graph is, the higher the degree of target points is, in general. Since the degree of a target point j is equal to the number of sensor nodes which observe j, we presume that the density of the extended communication graph is proportional to the lifetime of the *WSN* and, as a consequence, to the degree of challenge of the resulting *WSN* benchmark for an *OMT* solver.

5.3.2 Experiments and Results

These simulations ran on 3.60 GHz 8-core CPU with 8 GB memory. The CPU time limit was set to 1200 seconds and the memory limit to 3 GB. Parameters were set as follows: 10 sensor nodes, 4 target points, 2 critical points, a coverage with $K = 2$, an evasive constraint with $E = 2$ and a moving target constraint with $M = 1$.

Regarding *OMT* solvers, we chose the `OptiMathSAT` for our experiments, relying on our previous *(*see in Chapter 5.2.7) experiments which showed that the `OptiMathSAT` provided the most stable performance for different *WSN* constraints and parameter settings.

In terms of *graph density*, we investigate four different cases: two sparse (20%-30%, 40%-50%) and two dense groups (60%-70%, 80%-90%) of *WSN*s were generated. For each density group, we generated 20 *WSN*s. Table 5.5 summarizes the results of our experiments.

	Constraint setting	*#SAT/UNSAT*	*#TO*	*Optimum*	*Search steps*	*Time*	*Space*
20-30% density	all constraints on	12/8	0	1.0	1.0	1.4	123.4
	evasive off	12/8	0	1.0	1.0	2.5	112.6
	moving off	12/8	0	16.0	1.4	45.6	191.4
	evasive+moving off	12/8	0	37.0	6.6	35.9	157.0
40-50% density	all constraints on	19/0	1	30.5	1.2	173.9	430.6
	evasive off	20/0	0	35.5	1.9	51.0	293.3
	moving off	18/0	2	48.2	2.3	285.2	422.6
	evasive+moving off	18/0	2	53.0	6.8	273.4	374.9
60-70% density	all constraints on	16/0	4	78.8	2.5	327.2	613.0
	evasive off	19/0	1	80.2	3.3	190.7	405.0
	moving off	16/0	4	81.6	6.0	397.0	483.4
	evasive+moving off	18/0	2	83.9	8.4	522.7	625.8
80-90% density	all constraints on	20/0	0	119.3	3.5	192.8	678.9
	evasive off	19/0	1	120.8	8.3	190.5	457.9
	moving off	18/0	2	117.6	7.2	619.5	631.7
	evasive+moving off	14/0	6	116.8	8.2	794.5	847.2

Table 5.5: Summary of the results for *WSN*s of different graph density and different constraint settings.

Accordingly, we distinguish four constraint settings: (1) all the constraints are enabled, (2) the evasive constraint is disabled, (3) the moving target constraint is disabled, (4) both the evasive and moving target constraints are disabled. Table 5.5 shows the total number of solved *SAT/UNSAT* instances, as well as the number of timeouts (*#TO*). The *optimum* shows the average of the maximal lifetimes found for the SAT instances, and, in a similar manner, *Search steps* shows the average number of the linear search steps taken by the *OMT* solver to find the optimum. *Time* and *Space* represent the average runtime and memory consumption.

It is clearly visible in Table 5.5 that if the graph density is between 20-30%, almost half of the instances are *UNSAT*, i.e., some of the constraints are violated and there is no solution to the *WSN* problem. Even if there is a solution, the maximal lifetime is often only 1, i.e., the *WSN* can only survive for one time interval, which is not satisfying in real word application either. Interestingly, this situation seems to change slightly when disabling the moving target constraint, but one can observe real improvement in those numbers when the graph density reaches the 40%-50% interval.

Thus, we decided to drop the 20%-30% density group in the followings. For the remaining three density groups, the solver consumes increasing amount of memory as density grows, in the case of each constraint setting. The runtime follows a similar pattern, although it is easier for the solver to time out for some of the instances. The optimum steadily increases as well, which shows that a *WSN* consisting of the same amount of sensor nodes can survive for significantly longer if the density is higher. For the *OMT* solver, it requires more and more search steps to find the optimum, making the *WSN* instances of higher density more challenging to solve.

Summary We investigated how different constraint settings affect the difficulty of solving *WSN* benchmarks by *OMT* solvers. Naturally, we came to the logical conclusion that the more constraints we used and the higher parameter values we applied to them, the more challenging *WSN* benchmarks we got. Interestingly, our experiments show somehow the opposite, in terms of runtime and memory consumption as well as in terms of the optimal lifetime and the number of search steps. A significant increase of runtime occurs when the *moving target constraint* is switched off, mostly in the 60-70% and 80-90% density groups. The value of the maximal lifetime seems to be stabilized when both the *evasive and moving target constraints* are disabled. We get the most challenging instances when both the *evasive and moving target constraints* are switch off. Why does the loosening of constraints make the *WSN* optimization problem more difficult to solve? We think that the reason is the higher number of possible schedulings for the *WSN*. If the scheduler has more options to switch on and off certain sensor nodes in each and every time interval, the *OMT* solver needs to solve a more difficult combinatorial problem.

5.4 Puli a Problem-Specific *OMT* Solver

In Chapters 5.2 and 5.3, we investigated several aspects of applying *OMT* solvers to *WSN*s. In this section, we propose an idea of speeding up the *OMT* solving process by taking into consideration some resources in the systems and by applying regression analysis on those resource values. We show how to integrate this idea in search algorithms in the *OMT* framework and introduce a new *OMT* solver called `Puli`.

5.4.1 The Main Idea of `Puli`

Based on our preliminary plans, we did not aim to create a new *OMT* solver, but the previously described *OMT* solver could not be applied effectively to *WSN* problems. In the case of a *WSN* with a few 10 nodes, they often ran to timeout. Our goal was to make an *OMT* solver capable of optimizing a *WSN* of hundreds of nodes. During the earlier presented analyzes we observed that these *OMT* representations are designed to so-called, monotonous *OMT* problems. An *OMT* problem *monotonous* if, as we increment the value of the objective function, all the resulting *SMT* instances are *SAT* until exceeding the optimum. That is why there is no *UNSAT* instance below the optimum. For instance, the optimization of *WSN*s is a monotonous problem. For such problems, the assertion that (T) introduces can assign an exact value to the objective function: $f_{\text{OBJ}} = T$. Unlike the previously described *OMT* solvers, the `Puli` approaches the optimum by regression analysis from the models returned by *SMT* solver. The main idea was presented in [208].

5.4.2 The Algorithm of the `Puli`

In this section, we describe the algorithm of `Puli`. Why `Puli`? We were associated with `SYMBA` to *SIMBA (in the Lion King tale)*. The `Puli` is the Hungarian *SIMBA*. The `Puli` is a traditional Hungarian dog breed, the best known breeding dog in the world. In the first step, we examine the problem at lifetime $T = 1$, if the result is an *UNSAT* there is no solution. The randomly placed network is not suitable for observing the target*(s)*. If it is *SAT*, we can increase the lifetime and repeat the process again. This loop stops when we find the first *UNSAT (see Figure 5.6)* instance. The maximal lifetime corresponds to the *SAT* instance just before the first *UNSAT* one in the row. Of course, this is just a naive linear search which is not yet competitive with existing, more sophisticated *OMT* solving approaches. The problem is that we solve all the *SAT* instances for each possible T. To boost the search, we should not solve all the *SAT* instances if certain instances can safely be skipped. For this reason, we use regression analysis, in order to estimate at which T a *WSN* runs out of its resources and stops working.

An Example: In the following, we show our main idea through an example. As described earlier, we approach the optimum by regression analysis. In this example, the amount of network resources is 155 units. For this example, we distinguish three phases. In the first

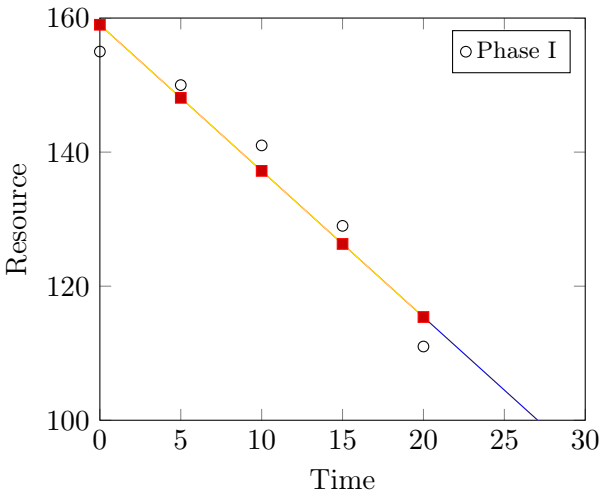


Figure 5.3: Phase I

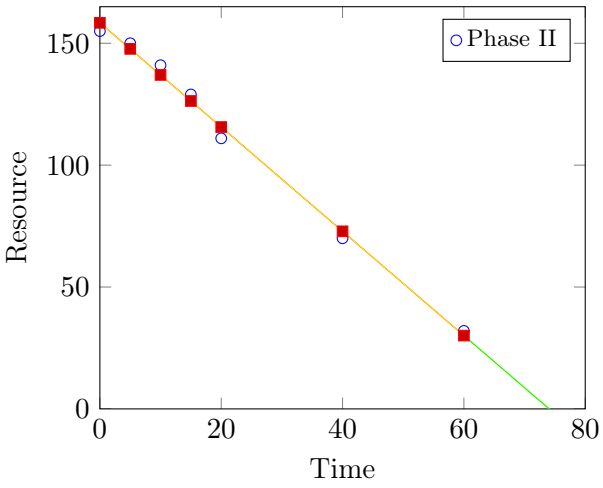


Figure 5.4: Phase II

phase, we examined the following lifetimes: $T = 1$, $T = 5$, $T = 10$, $T = 10$, $T = 15$, $T = 20$. After the first phase regression analysis *(see Figure 5.3)*, the estimated lifetime is 71.

Since the estimated optimum is much greater than 20, we can go on in greater steps *($T = 40$, $T = 60$)*. After the second phase regression analysis *(see Figure 5.4)*, the estimated lifetime is 72.

In the third phase, approaching the estimated optimum, we reduce the step interval again *($T = 65$, $T = 70$)*. After the third phase regression analysis *(see Figure 5.5)*, the estimated lifetime is 69. Since in $T = 70$ the result is already an *UNSAT*, we need to find the optimum between 65 and 70 *(eg. binary search)*. Figure 5.6 shows the complete regression analysis. The maximum lifetime in this network was 68.

Solver-specific Options: Our approach works for any *OMT* problem over *qfuflia* with a single objective function if the definition of a *resource function* f_{RES} is provided, in order

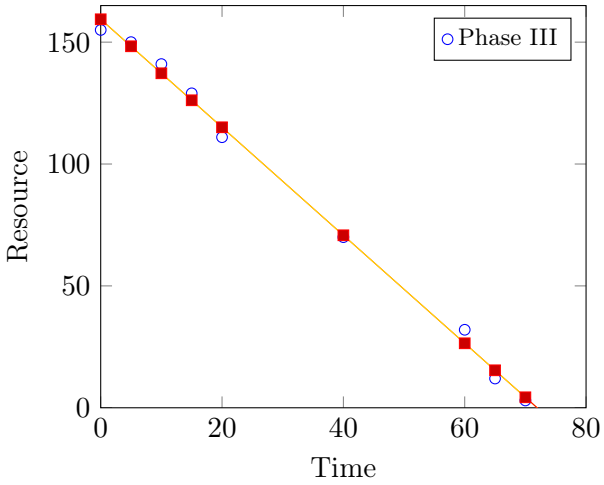


Figure 5.5: Phase III

to obtain data points $\big(T, f_{\text{RES}}(T)\big)$ for regression analysis where T is the current value for the objective function. For a *WSN*, the charge of the batteries of sensor nodes can be considered to be such a resource, which is continuously decreasing until draining. This is estimated by the following resource function:

$$\sum_{i=1}^{n} L_i - \sum_{i=1}^{n} \sum_{t=1}^{T} w_{i,t}$$

The value of the resource function is obtained by summing the lifetime of all the sensor nodes *(which is the theoretical maximum of the network's lifetime)*, and then subtracting the sum of the time intervals that have been used up so far. In fact, this is an estimation of how long the network will be operational from now on, i.e., how much charge is left in the batteries.

For our approach, it is also necessary to define a *resource target value* for the regression function, which we expect to be taken close to the optimum of the objective function. For a *WSN*, the resource target value is 0.

We introduce two solver-specific options to *SMT*-LIB: `opt-resource-fun` for defining the resource function and `opt-resource-target` for defining the resource target value. Both options can be interpreted by our solver, `Puli`. For the *WSN* problem, these options are specified as follows:

```
(define-fun resource-fun () Int
(-
(+ L0 L1 ...)
(+ (w 0 0) (w 0 1) ...)
))
(set-option :opt-resource-fun resource-fun)
(set-option :opt-resource-target 0)
```

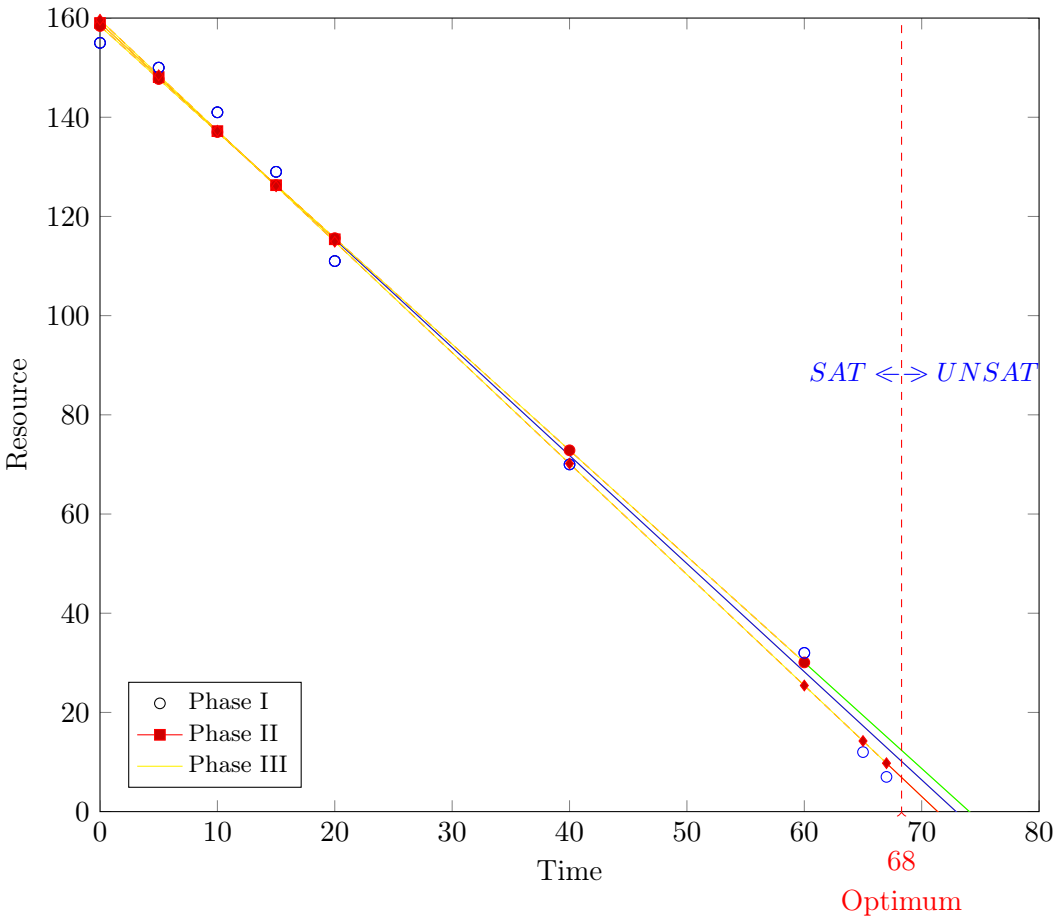


Figure 5.6: The Steps of the Regression Analysis

The `Puli`'s algorithm: Algorithm 11 shows how the boosted linear search works for maximization.[5] One must specify the lower bound lb for T as an input parameter. Different SMT-LIB instances are generated for different T values in a loop, an underlying SMT solver is called with those instances, and the results of those calls are saved in a pool of $(T, result)$ pairs, where $result \in \{SAT, UNSAT\}$. The functions SaveResult and LoadResult save and load data into/from this pool, respectively. If the current instance is SAT resp. $UNSAT$, then T is incremented resp. decremented, and a new iteration is about to start. There are two possible exits from the loop: (1) if the instance for lb is $UNSAT$, then there is no optimum; (2) if the instance for T is SAT and that for $T+1$ is $UNSAT$, then the optimum is T.

Otherwise, if the current instance is a SAT, the algorithm makes the SMT solver return the value $resource$ of the resource function and saves a new point $(T, resource)$ for regression analysis. If regression provides more than one root, we select the minimum root T' among all $T' > T$. Then we run a check on T and T' to decide (1) if it is worth to do a jump in the value of T instead of simply incrementing it, and (2) to what exact value to jump.

[5] From this, the algorithm for minimization can be obtained easily.

Some additional details about functions used in Algorithm 11:

GenerateSmtlib(T): An assertion on the value of the objective function f_{OBJ} is added. For maximization, this assertion is $f_{\text{OBJ}} \geq T$, by default. Previously introduced a special class of optimization problems for which this assertion can be more specific.

SmtSolve(*smtlib*): The prototype of `Puli` uses the `Z3` as the underlying *SMT* solver.

RegressionOnResourcePoints(): The prototype of `Puli` uses linear regression.

ConditionForJump(T,T'), **Jump**(T,T'): The prototype of `Puli` employs a simple jump strategy for the sake of skipping the most probable *SAT* instances and hitting the least possible *UNSAT* instances.

`Puli` is able to deal with monotonous and non-monotonous *OMT* problems as well. Dealing with *WSN* optimization as a monotonous problem speeds up `Puli`'s solving significantly, as our experiments show.

5.4.3 Experiments and Results

We ran experiments on the *WSN* benchmarks for four different constraint settings and three different density groups. Experiments were run on 3.6 GHz 8-core CPU with 8 GB memory. The wall clock time limit was set to 1200 seconds and the memory limit to 3 GB. Then we run `Puli`'s linear search boosted by regression and the solvers the `OptiMathSAT`, the `Z3` and the `Symba`.

Tables 5.6, 5.7 and 5.8 summarize the results of our experiments for the three different density groups, respectively. The columns show the total number of solved *SAT/UNSAT* instances, the number of timeouts ($\#TO$), the maximal optimum found for the SAT instances (*Optimum*), and the average runtime resp. memory consumption (*Time* resp. *Space*).

It is clearly visible that `Puli` provides remarkably stable performance: it can solve almost all instances with significantly low runtime and less memory. Compared to the `OptiMathSAT`, the `Z3` and the `Symba` are worth to use on the easier benchmarks, i.e., the ones with lower density and fewer constraints, but their performance significantly declines as benchmarks getting harder.

	Constraint settings	*#SAT/UNSAT*	*#TO*	*Optimum*	*Runtime*	*Space*
	all constraints on	19/0	1	30.5	63.5	75.2
`Puli`'s	evasive off	20/0	0	35.5	3.5	66.45
	moving off	20/0	0	48.2	6.4	72.6
	evasive+moving off	20/0	0	53.0	7.95	63.65
	all constraints on	19/0	1	30.5	173.9	430.6
`OptiMathSAT`	evasive off	20/0	0	35.5	51.0	293.3
	moving off	18/0	2	48.2	285.2	422.6
	evasive+moving off	18/0	2	53.0	273.4	374.9
	all constraints on	10/0	10	30.5	605.85	554.4
`Z3`	evasive off	20/0	0	35.5	64.2	388.35
	moving off	16/0	4	48.2	293	614.4
	evasive+moving off	20/0	0	53.0	8.75	138.75
	all constraints on	10/0	10	30.5	654.5	632.9
`Symba`	evasive off	20/0	0	35.5	152.5	421.45
	moving off	12/0	8	48.2	632.05	644.4
	evasive+moving off	20/0	0	53.0	118.1	187.85

Table 5.6: Results for different constraint settings for *WSN*s of 40-50% density.

	Constraint settings	*#SAT/UNSAT*	*#TO*	*Optimum*	*Runtime*	*Space*
	all constraints on	19/0	1	78.8	79.95	122.32
`Puli`'s linear search	evasive off	20/0	0	80.2	14.5	86.4
	moving off	20/0	0	81.6	44.95	102.75
	evasive+moving off	20/0	0	83.9	23.75	69.9
	all constraints on	16/0	4	78.8	327.2	613.0
`OptiMathSAT`	evasive off	19/0	1	80.2	190.7	405.0
	moving off	16/0	4	81.6	397.0	483.4
	evasive+moving off	18/0	2	83.9	522.7	625.8
	all constraints on	6/0	14	30.5	858.85	1025.6
`Z3`	evasive off	16/0	4	35.5	325.15	592.35
	moving off	13/0	7	48.2	525.5	672
	evasive+moving off	20/0	0	53.0	11.95	143.35
	all constraints on	7/0	13	30.5	924.55	898.05
`Symba`	evasive off	15/0	5	35.5	572.3	731.45
	moving off	9/0	11	48.2	794.7	921.6
	evasive+moving off	20/0	0	53.0	127.8	197.25

Table 5.7: Results for different constraint settings for *WSN*s of 60-70% density.

Algorithm 11 `Puli`'s algorithm for maximization

```
1: procedure MAXIMIZATION(lb)
2:     T ← lb
3:     while true do
4:         if LOADRESULT(T) = SAT then
5:             increment T
6:         else if LOADRESULT(T) = UNSAT then
7:             if T = lb then
8:                 return null
9:             end if
10:            decrement T
11:        else
12:            smtlib ← GENERATESMTLIB(T)
13:            (T,result,resource) ← SMTSOLVE(smtlib)
14:            SAVERESULT(T, result)
15:            if result = SAT then
16:                if LOADRESULT(T + 1) = UNSAT then
17:                    return T
18:                end if
19:                SAVERESOURCEPOINT(T, resource)
20:                regression ← REGRESSIONONRESOURCEPOINTS()
21:                T′ ← minimum root of regression where T′ > T
22:                if CONDITIONFORJUMP(T,T′) then
23:                    T ← JUMP(T,T′)
24:                end if
25:            end if
26:        end if
27:    end while
28: end procedure
```

5.5 Conclusions

In this chapter, we investigated a *WSN* optimization problem: how to maximize the lifetime of the *WSN* while not violating certain dependability and security constraints. Here is, three such constraints are addressed: the K-coverage constraint, the evasive constraint and the moving target constraint. Most of the existing approaches focus on the maximization problem only in combination with K-coverage, while others focus on using multiple constraints, but not addressing the maximization problem at all. By using *OMT* formalism, it is possible to combine all the aforementioned constraints as well as to specify an optimization objective.

We looked into certain graph properties that can cause the aforementioned phenomena. First of all, we defined the *extended communication graph* for a *WSN*, and then we investigated how the *density* of this graphs affects the maximal lifetime of the *WSN*, the

runtime and the number of search steps taken by the *OMT* solver. Our experiments show that the density of 40-50% is minimally required for getting a reasonable lifetime for a *WSN*. Furthermore, the density of 80-90% makes a *WSN* benchmark challenging enough for the *OMT* solver *OptiMathSAT*. We discovered an interesting phenomenon as well: by loosening the *WSN* constraints, *WSN* benchmarks get easier to solve, especially when the moving target constraint is disabled. This shows that the optimization process dominates the runtime as opposed to the *SMT* solving process. We made the resulting *OMT* benchmarks and log files publicly available, as well as the scripts we used. Although we used fixed parameters for all the *WSN* constraints, we presume that *WSN* benchmarks become even easier to solve when increasing the value of the constraint parameters, such as the coverage parameter K, or even the parameters E and M for the evasive and the moving target constraints, respectively. Increasing the ratio of critical points in a *WSN* could have a similar effect. We are planning to do further experiments in this direction, as part of future work.

Finally, we proposed the idea of speeding up the *OMT* solving process by taking into consideration a *resource function* in the system to optimize and by applying *regression analysis* on those resource values. Furthermore, we introduced a class of *OMT* problems, the so-called *monotonous* problems, for which the solving process can be further boosted. We introduced a *new OMT solver* called `Puli` and reported experiments on different *OMT* benchmarks for *WSNs*. The results show that `Puli` significantly outperforms the *OMT* solvers `OptiMathSAT`, `Z3` and `Symba` on those benchmarks. In general, `Puli` can solve any qfuflia problem with a single objective function and can apply its the regression-based boosting if the definition of a *resource function* is provided. For this, we introduced two solver-specific options to *SMT-LIB*: `opt-resource-fun` for defining the resource function and `opt-resource-target` for defining the resource target value. Both options can be interpreted by `Puli`.

6 Part IV

In this chapter, we introduce 6 novel k-hop environments defined 3 density-based and 3 redundancy-based metrics: Weighted Communication Graph Density ($\mathcal{WCGD}^{[k]}$), Relative Communication Graph Density ($\mathcal{RCGD}^{[k]}$), Weighted Relative Communication Graph Density ($\mathcal{WRCGD}^{[k]}$), Communication Graph Redundancy (${\mathcal{CGR}^{[k]}}_{vb}$) - *(clique value-based)*, Weighted Communication Graph Redundancy (${\mathcal{WCGR}^{[k]}}_{sb}$) -*(clique sized-based)* and Weighted Communication Graph Redundancy (${\mathcal{WCGR}^{[k]}}_{vb}$) - *(clique value-based)*. We compare them to known graph metrics, and show that they can be used for node ranking.

6.1 Introduction

Node localization and ranking are essential issues in wireless sensor networks (*WSNs*). We model *WSNs* by communication graphs. In our interpretation a communication graph can be either directed, in the case of heterogeneous sensor nodes or undirected, in the case of homogeneous sensor nodes, and must be strongly connected. There are many metrics to characterize networks, most of them are either global ones or local ones. The local ones consider only the immediate neighbors of the observed nodes. So our main goal was to construct metrics that interpret the local properties of the nodes in a wider environment. For example, how dense the environment of the given node, or in which extent it can be relieved within its environment.

In this dissertation, the metrics are defined for weighted directed graphs (generally), but we assume that the communication graph is strongly connected, the cost of communication between each node is constant *(we do not use weights)*, and the network consists of homogeneous nodes.

To test the new metrics we used our own representation [198] and *SAT* solvers [203, 205].

6.2 Preliminaries

Given a randomly-deployed sensor network with homogeneous or heterogeneous nodes. Also given a mapping that shows which sensor is able to communicate with which sensors directly. Accordingly, the communication graph is a weighted directed graph $\mathcal{D} = (\mathcal{S}; \mathcal{E}_{\mathcal{C}}, \mathcal{W})$, where $\mathcal{S}$ is the set of nodes, which represents the sensors, $\mathcal{E}_{\mathcal{C}} \subseteq \mathcal{S} \times \mathcal{S}$ is the set of edges, and $\mathcal{W}$ is the set of weights. An edge $(x_i, x_j) \in \mathcal{E}_{\mathcal{C}}$ represents the possibility of messaging from node x_i to x_j in $\mathcal{D}$, i.e., the sensor represented by x_j is in the transmission range of

x_i. The $\mathcal{W}_{ij}$ denotes the communication costs of the (x_i, x_j) message.

In an extended model, communication costs can be different in the future. The weight of the edges can be, for example, based on the distance of nodes, the encryption levels, the observed terrain, the sensitivity of the channel, etc.

In the case of homogeneous sensors, the $\mathcal{D}$ graph is symmetric, accordingly $\mathcal{D} = (\mathcal{S}; \mathcal{E}_\mathcal{C}, \mathcal{W})$ is equivalent to a simple weighted undirected graph $\mathcal{G} = (\mathcal{S}; \mathcal{E}_\mathcal{C}, \mathcal{W})$. In case of an weighted undirected graph $\mathcal{G} = (\mathcal{S}; \mathcal{E}_\mathcal{C}, \mathcal{W})$, we define a clique as a subset of the nodes set $\mathcal{C}l \subseteq \mathcal{S}$, such that for every two nodes in $\mathcal{C}l$, there exists an edge connecting them. The weight of a $\mathcal{C}l$ is the sum of the weight of their edges.

If our communication graph is directed, we define a clique as a subset of the node set $\mathcal{C}l \subseteq \mathcal{S}$, such that for every two nodes in $\mathcal{C}l$, there exists an edge from the first one to the second one, and from the second one back to the first one. The weight of the $\mathcal{C}l$ is defined as above, considering that $\mathcal{W}_{ij}$ and $\mathcal{W}_{ji}$ are not necessarily equal. A maximal clique is a clique which is not a proper subset of any other clique. A n-clique is a clique which contains exactly n vertices.

6.3 k-hop-based Graph Density and Redundancy Metrics

In this section, we present some spanning tree and clique-based graph density metrics. With spanning tree-based metrics, we define graph density, whereas clique-based redundancy metrics mean the degree of relieving in our interpretation. We use the notion of k-hop environment of a node u, denoted by $\mathcal{G}^{[k]}(u)$, which is a subgraph of graph $\mathcal{G}$, which contains u and the nodes which can be reached from u from a path, which length is smaller than or equal to k, and which contains edges between these nodes from $\mathcal{G}$. We compute local metrics for u by computing a graph metrics for $\mathcal{G}^{[k]}(u)$. The parameter k should be a relatively small number otherwise $\mathcal{G}^{[k]}(u)$ could be the whole graph. The metrics over the k-hop environment of a node can characterize the node more properly than considering merely the node itself. On the other hand, these metrics characterize not only the node but its environment.

Taking into account the constraints mentioned earlier, the basic notations are:

- u: the candidate node;
- k: the number of hops;
- $\mathcal{N}$, $\mathcal{V}$: the number of nodes and edges of graph $\mathcal{G}$;
- $\mathcal{N}^{[k]}(u)$,$\mathcal{V}^{[k]}(u)$: the number of nodes and edges of graph $\mathcal{G}^{[k]}(u)$;
- $\mathcal{C}l$, $\mathcal{M}$: the set of maximum cliques of graph $\mathcal{G}$ and the cardinality of this set;

- $\mathcal{C}l^{[k]}(u)$, $\mathcal{M}^{[k]}(u)$: the set of maximal cliques of graph $\mathcal{G}^{[k]}(u)$ and the cardinality of this set;
- $\mathcal{T}^{[k]}(u)$, $\mathcal{T}$: the number of edges of the minimum cost spanning tree of graph $\mathcal{G}^{[k]}(u)$ and $\mathcal{G}$. Note, that in case of a communication graph we have that $\mathcal{T} = \mathcal{N} - 1$, regardless whether the graph is directed or undirected;
- s: the spreading factor, which is rather a technical value to enlarge small differences in the metrics, in this dissertation we set $s = 2.71$;
- cs: the minimum clique size that can be considered by the metrics *(minimum value is 2)*.

6.3.1 Spanning Tree-based Metrics

Spanning tree-based approaches can be found in the wide area of network protocols. For example, a known technique is Time-To-Live *(TTL)*. It works as follows: routing methods try to find the best path for forwarding the collected data, the TTL mechanism is used to limit the number of hops to avoid over-overlapping of paths and to balance the data load on the nodes and the energy consumption [156]. They use also small k values.

We define the graph density of the graphs $\mathcal{G}$ and $\mathcal{G}^{[k]}(u)$ as follows:

$$\mathcal{GD} = \frac{\mathcal{V}}{\mathcal{T}} \tag{6.3.1}$$

$$\mathcal{GD}^{[k]}(u) = \frac{\mathcal{V}^{[k]}(u)}{\mathcal{T}^{[k]}(u)} \tag{6.3.2}$$

The graph density takes its maximum if the graph is complete. In the case of undirected graphs the maximum is: $\frac{\mathcal{N}(\mathcal{N}-1)}{2(\mathcal{N}-1)} = \frac{\mathcal{N}}{2}$. In the case of directed graphs the maximum is: $\frac{\mathcal{N}(\mathcal{N}-1)}{\mathcal{N}-1} = \mathcal{N}$. The graph density takes its maximum if the graph is a tree. In the case of undirected graphs the minimum is: $\frac{\mathcal{N}-1}{\mathcal{N}-1} = 1$, since the graph is a communication graph, i.e., it is strongly connected. If the graph is directed, then the minimum is: $\frac{2(\mathcal{N}-1)}{\mathcal{N}-1} = 2$, because of the same reason.

Communication and Weighted Communication Graph Density

We define the communication graph density of node u in its k-hop environment as follows:

$$\mathcal{CGD}^{[k]}(u) = s^{\frac{\mathcal{V}^{[k]}(u)}{\mathcal{T}^{[k]}(u)}} \tag{6.3.3}$$

The $\mathcal{CGD}^{[k]}(u)$ can be used also as a local metric for a node, and computed quickly for all nodes and use to rank them.

We define the weighted communication graph density of node u in its k-hop environment as follows:

$$\mathcal{WCGD}^{[k]}(u) = s^{\frac{\mathcal{V}^{[k]}(u)}{\mathcal{T}^{[k]}(u)}} \frac{\mathcal{N}^{[k]}(u)}{\mathcal{N}} \tag{6.3.4}$$

The $\mathcal{WCGD}^{[k]}(u)$ is no longer a purely local metric, but takes into account the number of nodes in the k-hop environment.

Relative Communication Graph Density

We define the relative communication graph density of node u in its k-hop environment as follows:

$$\mathcal{RCGD}^{[k]}(u) = s^{\frac{\mathcal{GD}^{[k]}(u)}{\mathcal{GD}}} = s^{\frac{\mathcal{V}^{[k]}(u)\mathcal{T}}{\mathcal{T}^{[k]}(u)\mathcal{V}}} \tag{6.3.5}$$

It maximizes its value when the k-hop environment of u, i.e., $\mathcal{G}^{[k]}(u)$ is a complete graph and the rest of the graph is a tree or consists of several trees.
The minimum is - vice versa - assumes that the k-hop environment of u is a tree and the rest of the graph is a complete graph.
If we consider the two extremes, i.e., if the communication graph is a complete graph or if it is a tree, interestingly enough, we get the same relative communication graph density, which is s. If the communication graph is a complete graph, then for any $k >= 1$ and for any node u we have that $\mathcal{G}^{[k]}(u)$ is equal to $\mathcal{G}$, so, $\frac{\mathcal{V}^{[k]}(u)}{\mathcal{T}^{[k]}(u)} = \frac{\mathcal{V}}{\mathcal{T}}$,i.e., $\frac{\mathcal{V}^{[k]}(u)\mathcal{T}}{\mathcal{T}^{[k]}(u)\mathcal{V}} = 1$. On the other hand, if the communication graph is a tree, then its communication graph density is a constant (1 if the graph is undirected, 2 if it is directed) for any n and u, so again $\frac{\mathcal{V}^{[k]}(u)\mathcal{T}}{\mathcal{T}^{[k]}(u)\mathcal{V}} = 1$.
We get the same result for the two extreme cases, because this metric shows the relative density of the subgraph related to the whole graph. A tree has a very small density, and a complete graph has a very high density, but if we take a subgraph of a tree then it has the same density as the whole, and the same is true for a complete graph. So they have the same relative density.
This metric shows whether the k-hop environment of a node is denser than the whole graph, or has the same density, or it is less dense. This means that if

- $\mathcal{RCGD}^{[k]}(u) = s$, then $\mathcal{G}^{[k]}(u)$ has the same CGD as $\mathcal{G}$;
- $\mathcal{RCGD}^{[k]}(u) < s$, then $\mathcal{G}^{[k]}(u)$ has smaller CGD than $\mathcal{G}$;
- $\mathcal{RCGD}^{[k]}(u) > s$, then $\mathcal{G}^{[k]}(u)$ has bigger CGD than $\mathcal{G}$;

where CGD means communication graph density.
Note, that this metric is computed by dividing a local property by a global one.

Weighted Relative Communication Graph Density

We define the weighted relative communication graph density of node u in its k-hop environment as follows:

$$\mathcal{WRCGD}^{[k]}(u) = \mathcal{RCGD}^{[k]}(u)\frac{\mathcal{N}^{[k]}(u)}{\mathcal{N}} = s\frac{\mathcal{V}^{[k]}(u)\mathcal{T}}{\mathcal{T}^{[k]}(u)\mathcal{V}}\frac{\mathcal{N}^{[k]}(u)}{\mathcal{N}} \tag{6.3.6}$$

Note, that this metric is computed as a multiplication of two numbers, which are both computed by dividing a local property by a global one, so we have $(local'/global') * (local''/global'')$.
This metric also takes in consideration how many nodes in the n-hop environment of the node are u. A node is more valuable if its k-hope environment is bigger.

6.3.2 Clique-based Metrics

During the work of a *WSN* the topology of the network may change because some sensors may go wrong, or the transmission range can be less. If a node can be found in a dense *(redundant)* environment then it may happen more often that communication interference occurs and the routing consumes more resources; on the other hand, the environment itself is more fault tolerant. In a sparse environment, routing is easier, communication interference is less frequent, but the environment is less fault tolerant. The aim of topology control techniques is to reduce the cost of the distributed algorithms interpreted on the network. But the network-quality characteristics *(like scalability, coverage, fault tolerance, etc.)* must not fall below a required level. A clique is a complete subgraph, so they have high communication redundancy, on the other hand they allow high fault tolerance, results in high coverage, etc.
First of all, we define the average clique size as follows:

$$\overline{\mathcal{CL}} = \frac{1}{\mathcal{M}}\sum_{i=1}^{\mathcal{M}} |\mathcal{C}l_i|_{>=cs} \tag{6.3.7}$$

The average clique size is maximal, if the graph is complete. Its minimum is cs if all maximal cliques have the size cs. It is not defined if there is no clique with the size at least cs. Its maximum is $\mathcal{N}$ if the communication graph is complete, because then we have only one maximal clique, the graph itself. The clique problem, the problem of finding all maximal size cliques, is a well-known *NP*-complete problem. It means that is not feasible to find all maximal cliques in a large graph. So we can not use clique based metrics to guide topology control techniques, except if we work with relatively small graphs, like in the k-hop environment of a node.
Remark: The average clique size of node u within k-hop environment is donated as follows: $\overline{\mathcal{CL}}^{[k]}(u)$.

Clique size-based metrics

So we define the clique size-based communication graph redundancy of node u within k-hop environment as follows:

$$\mathcal{CGR}_{sb}{}^{[k]}(u) = \frac{1}{\mathcal{M}^{[k]}(u)} \sum_{i=1}^{\mathcal{M}^{[k]}(u)} \left|\mathcal{C}l^{[k]}(u)_i\right|_{>=cs} \tag{6.3.8}$$

It only shows the average clique size within k-hop environment of node u, but it ignores the number of nodes within the k-hop environment.

We define weighted communication graph redundancy of node u within k-hop environment as follows:

$$\mathcal{WCGR}_{sb}{}^{[k]}(u) = \mathcal{CGR}_{sb}{}^{[k]}(u)\frac{\mathcal{N}^{[k]}(u)}{\mathcal{N}} \tag{6.3.9}$$

This metric uses also the number of nodes. This can be considered to be a local metric, because computationally intensive tasks *(find cliques)* typically occur in a k-hop environment.

Clique value-based metrics

Since a 4-clique is more valuable in a graph consisting of 6 nodes than in a graph with 100 nodes, we shall take into consideration the number of nodes in the graph, which is denoted by $\mathcal{N}$, to compute the value of a clique. We also use the average clique size to normalize this value.

So we define the value of a clique as follows:

$$\mathcal{CL}_V = \frac{|\mathcal{C}l|_{>=cs}}{\mathcal{N}} s^{|\mathcal{C}l|_{>=cs}/\overline{\mathcal{CL}}} \tag{6.3.10}$$

We define also the average value of cliques as follows:

$$\overline{\mathcal{CL}}_V = \frac{1}{\mathcal{M}} \sum_{i=1}^{\mathcal{M}} \frac{|\mathcal{C}l_i|_{>=cs}}{\mathcal{N}} s^{|\mathcal{C}l_i|_{>=cs}/\overline{\mathcal{CL}}} \tag{6.3.11}$$

We define also the average value of cliques within the k-hop environment, also called clique value-based communication graph redundancy as follows:

$$\mathcal{CGR}_{vb}{}^{[k]}(u) = \frac{1}{\frac{1}{\mathcal{M}^{[k]}(u)} \sum_{i=1}^{\mathcal{M}^{[k]}(u)} \frac{|\mathcal{C}l^{[k]}(u)_i|_{>=cs}}{\mathcal{N}^{[k]}(u)} s^{\frac{|\mathcal{C}l^{[k]}(u)_i|_{>=cs}}{\overline{\mathcal{CL}}^{[k]}(u)}}} \tag{6.3.12}$$

This metric is the pair of $\mathcal{CL}_V$ in case of $\mathcal{G}^{[k]}(u)$. This is a local metric, but this notion does not take into consideration the number of nodes in the k-hop environment of u. Without reciprocal, the peripheral but relievable nodes are ranked in advance.

After considerating the number of nodes in the k-hop and conversion, we define weighted clique value-based communication graph redundancy as follows:

$$\mathcal{WCGR}_{vb}{}^{[k]}(u) = \frac{1}{\mathcal{CGR}_{vb}{}^{[k]}(u)} \frac{\mathcal{N}^{[k]}(u)}{\mathcal{N}} = \tag{6.3.13}$$

$$= \frac{1}{\mathcal{M}^{[k]}(u)} \sum_{i=1}^{\mathcal{M}^{[k]}(u)} \frac{\mathcal{N}^{[k]}(u)|\mathcal{C}l^{[k]}(u)_i|_{>=cs}}{\mathcal{N}^2} s^{\frac{|\mathcal{C}l^{[k]}(u)_i|_{>=cs}}{\overline{\mathcal{CL}}^{[k]}(u)}} \tag{6.3.14}$$

This metric is the pair of $\overline{\mathcal{CL}_V}$ in case of $\mathcal{G}^{[k]}(u)$.

6.4 Comparisons with Other Metrics

In this chapter, we considered networks with 200-500 nodes at 15-40 % densities. The k value in each case is less than 3. An important constraint was that the largest k-hop environment must be smaller than the quarter of a complete graph.

For simulating and analyzing networks we used a self-developed *Python* tool *based on NetworkX* [1]. For computing pairwise correlation of metrics we used *pandas* [2]. Many metrics are only implemented for undirected graphs in *NetworkX*, therefore, the comparisons were done only on undirected graphs. The results of the correlation analysis presented in Tables 6.1 - 6.3 *(average values of 1000 runs)*, show some interesting phenomena and experience. The abbrevation c. means centrality.

1-hop based environment

	k-hop based metrics					
	$\mathcal{WCGD}^{[k]}$	$\mathcal{RCGD}^{[k]}$	$\mathcal{WRCGD}^{[k]}$	$\mathcal{WCGR}_{sb}{}^{[k]}$	$\mathcal{CGR}_{vb}{}^{[k]}$	$\mathcal{WCGR}_{vb}{}^{[k]}$
Clustering coeff.	0,06	0,19	0,00	0,37	-0,17	0,14
Eccentricity	-0,07	-0,21	-0,24	-0,15	-0,31	-0,21
Betweenness c.	-0,04	-0,05	0,06	-0,12	0,22	-0,01
Degree c.	**0,54**	**0,87**	**0,94**	**0,76**	**0,84**	**0,83**
Closeness c.	0,05	0,23	0,28	0,17	0,37	0,23
Eigenvector c.	**0,67**	**0,61**	**0,64**	0,49	0,28	**0,68**

Table 6.1: Correlations with other metrics where k is 1

It can be seen from the Table 6.1 that within 1-hop environment the defined metrics show their most significant correlation with degree centrality. The correlation is the strongest between $\mathcal{WRCGD}$ and degree centrality, the correlation is over 90 %. $\mathcal{WCGD}$, $\mathcal{RCGD}$, $\mathcal{WRCGD}$ and $\mathcal{WCGR}_{vb}$ metrics are also strongly correlated with the eigenvector centrality.

[1] https://networkx.github.io/
[2] https://pandas.pydata.org

2-hop based environment

	k-hop based metrics					
	$\mathcal{WCGD}^{[k]}$	$\mathcal{RCGD}^{[k]}$	$\mathcal{WRCGD}^{[k]}$	${\mathcal{WCGR}_{sb}}^{[k]}$	${\mathcal{CGR}_{vb}}^{[k]}$	${\mathcal{WCGR}_{vb}}^{[k]}$
Clustering coeff.	-0,07	0,04	-0,15	0,08	-0,26	-0,06
Eccentricity	-0,15	-0,24	-0,38	-0,22	-0,45	-0,33
Betweenness c.	0,06	0,03	0,22	0,05	0,32	0,17
Degree c.	**0,54**	**0,78**	**0,83**	**0,72**	**0,67**	**0,72**
Closeness c.	0,08	0,26	0,44	0,24	0,53	0,36
Eigenvector c.	**0,87**	**0,68**	**0,67**	**0,57**	0,26	**0,71**

Table 6.2: Correlations with other metrics, where k is 2

It can be seen from the Table 6.2 that the defined metrics within 2-hop environment showed a weaker correlation with the degree centrality and stronger with the eigenvector centrality, since the degree of neighbors of the examined node also affects the density and redundancy of the environment. The strongest correlation with the degree centrality is still shown with $\mathcal{WRCGD}$, while with the eigenvector centrality correlates best with $\mathcal{WCGD}$.

3-hop based environment

The correlations in the 3-hop environment are shown in the Table 6.3 . In general, the correlations with the degree centrality and the eigenvector centrality are no longer significant, the eccentricity and the closeness centrally correlations are reinforced.

	k-hop based metrics					
	$\mathcal{WCGD}^{[k]}$	$\mathcal{RCGD}^{[k]}$	$\mathcal{WRCGD}^{[k]}$	${\mathcal{WCGR}_{sb}}^{[k]}$	${\mathcal{CGR}_{vb}}^{[k]}$	${\mathcal{WCGR}_{vb}}^{[k]}$
Clustering coeff.	0,12	0,03	-0,20	0,03	-0,24	-0,19
Eccentricity	-0,27	-0,18	**-0,55**	-0,30	**-0,56**	**-0,51**
Betweenness c.	0,18	0,07	0,36	0,14	0,38	0,29
Degree c.	0,46	**0,55**	**0,63**	**0,56**	**0,54**	**0,62**
Closeness c.	0,38	0,25	**0,63**	0,34	**0,67**	**0,54**
Eigenvector c.	**0,72**	**0,55**	**0,52**	**0,52**	-0,26	**0,59**

Table 6.3: Correlations with other metrics, where k is 3

In the following, we analyze the relationship between the new and already known metrics in detail.

Weighted Communication Graph Density

The metric $\mathcal{WCGD}^{[k]}(u)$ correlates strongly with eigenvector centrality. There is a not too strong but significant correlation with degree centrality also, and there is no relevant correlation with other metrics. If we want to characterize this metric on the basis of the above, then a high $\mathcal{WCGD}^{[k]}(u)$ value node has the following properties (in k-hop environment, if $k = 3$):

- average probability of high number of direct connections,
- high probability of high degree of neighbors,
- weak probability of central location.

Relative Communication Graph Density

The metric $\mathcal{RCGD}^{[k]}(u)$ has average correlation with degree centrality and eigenvector centrality, weak but significant contact with closeness centrality, and there is no relevant correlation with other metrics. So a high $\mathcal{RCGD}^{[k]}(u)$ value node has the following properties (in k-hop environment, if $k = 3$):

- average probability of high number of direct connections,
- average probability of high degree of neighbors.

Weighted Relative Communication Graph Density

The metric $\mathcal{WRCGD}^{[k]}(u)$ has an average linear correlation with degree centrality, closeness centrality, and eigenvector centrality, and suggests a weak correlation with betweenness centrality, but with eccentricity shows an average but inverse correlation. So a high $\mathcal{WRCGD}^{[k]}(u)$ value node has the following properties (in k-hop environment, if $k = 3$):

- average probability of high number of direct connections,
- average probability of central location,
- average probability of high degree of neighbors,
- weak probability of high geodesic distance from any other node.

Weighted Communication Graph Redundancy *(size-based)*

The metric $\mathcal{WCGR}_{sb}{}^{[k]}(u)$ has average correlation with degree centrality and eigenvector centrality. It suggests a weak but significant correlation with closeness centrality and inverse correlation with eccenticity. There is no relevant correlation with other metrics. So a high $\mathcal{WCGR}_{sb}^{[k]}(u)$ value node has the following properties (in k-hop environment, if $k = 3$):

- average probability of high number of direct connections,

- average probability of high degree of neighbors,
- weak probability of central location.

Communication Graph Redundancy *(value-based)*

The metric $\mathcal{CGR}_{vb}^{[k]}(u)$ has an average linear correlation with degree centrality and closeness centrality. It suggests a weak correlation with betweenness centrality. It shows an average but inverse correlation with eccentricity. So a high $\mathcal{CGR}_{vb}^{[k]}(u)$ value node has the following properties (in k-hop environment, if $k = 3$):

- average probability of high number of direct connections,
- average probability of central location,
- weak probability of great geodesic distance from any other node.

Weighted Communication Graph Redundancy *(value-based)*

The metric $\mathcal{WCGR}_{vb}^{[k]}(u)$ has an average linear correlation with degree centrality, closeness centrality and eigenvector centrality. It suggests a weak correlation with betweenness centrality, but with eccentricity shows an average but inverse correlation. So a high $\mathcal{WCGR}_{vb}^{[k]}(u)$ value node has the following properties (in k-hop environment, if $k = 3$):

- average probability of high number of direct connections,
- average probability of high degree of neighbors,
- average probability of central location,
- weak probability of great geodesic distance from any other node.

6.5 Node Ranking

In this section, we show how to use the different metrics to make node ranking *(top* 30 *selection)*. The generated network (shown in Figure 6.1) contains 100 randomly deployed and homogeneous sensor nodes *(vertices)* with 926 connections *(edges)*. The density is 18.7% , the transmission range is 55 m, the area is 300 m x 300 m and the k-hop number is 3. The communication graph of the exemplary network are shown Figure 6.2.

Both the spanning tree and the clique based metrics show the denser environments of the network. Since this network consists of only 100 nodes it does not give a real picture of the metrics, but we can still see the tendencies.

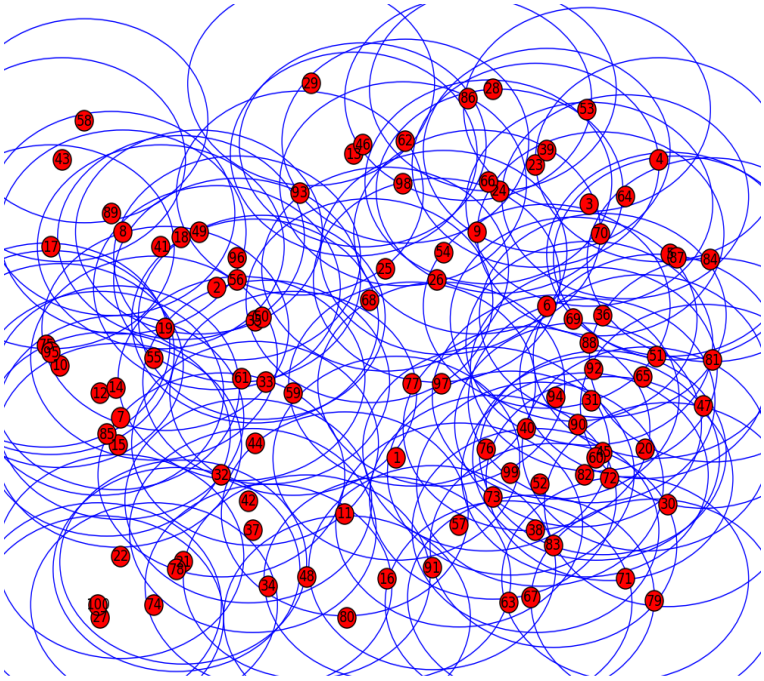


Figure 6.1: Randomly Deployed Sensor Network

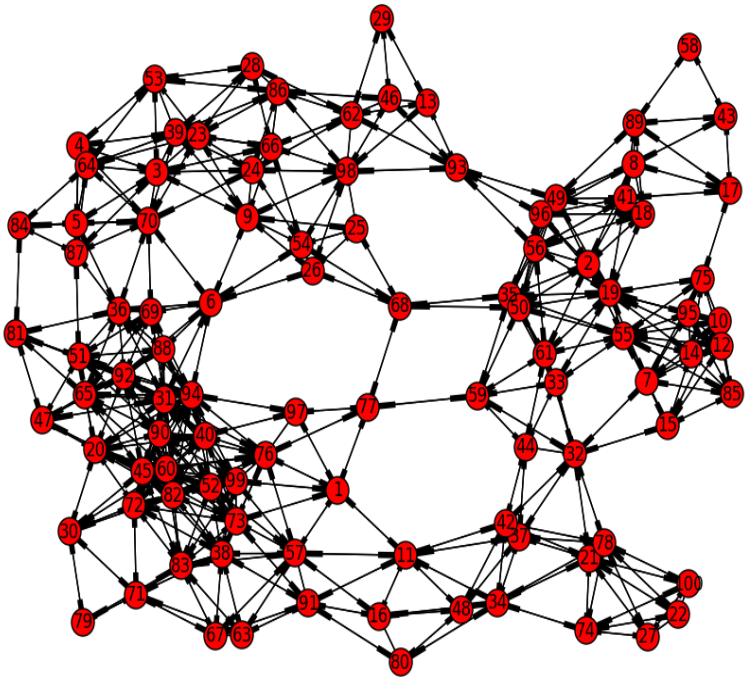


Figure 6.2: Communication Graph

Spanning tree-based metrics (density)

Figures 6.3 - 6.5 show how to use the spanning tree based metrics to make ranking nodes. The task of all three metrics is to designate the densest environments. Based on the

comparisons, the most significant feature of $\mathcal{WCGD}$ is that the neighbors and the neighbors' neighbors have a high degree. Figure 6.3 shows that the top 3 node and at least 40 % of the selected nodes are centrally located. The metric $\mathcal{RCGD}$ showed no significant correlation with the closeness centrality and eccentricity. In Figure 6.4 we can see that among the selected nodes there are only few nodes in central position. The metric $\mathcal{RCGD}$ primarily marks the nodes within the densest areas. The metric $\mathcal{WCGD}$ selects those nodes whose density is high within their k-hop environment and centrally located. Figure 6.5 shows that the top 3 node and at least 80 % of the selected nodes are centrally located.

In these figures, we use the following colour codes: top 1 rank node is red, top 2 is green, top 3 is yellow, top $4-10$ are blue, top $11-20$ are pink, top $21-30$ are orange, the rest is cyan.

Figure 6.3 shows the top 30 ranked nodes based on the *Weighted Communication Graph Density* metric. In 3-hop environments, the highest weighted communication graph density has nodes 77, 68 and 26.

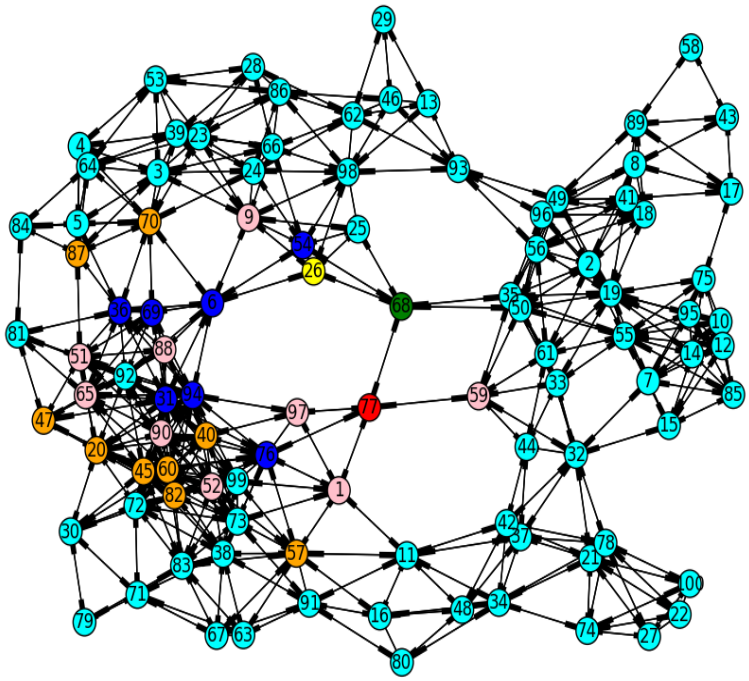


Figure 6.3: Weighted Communication Graph Density

Figure 6.4 shows the top 30 ranked nodes based on the *Relative Communication Graph Density* metric. In 3-hop environments, the highest relative communication graph density has nodes 30, 79 and 71.

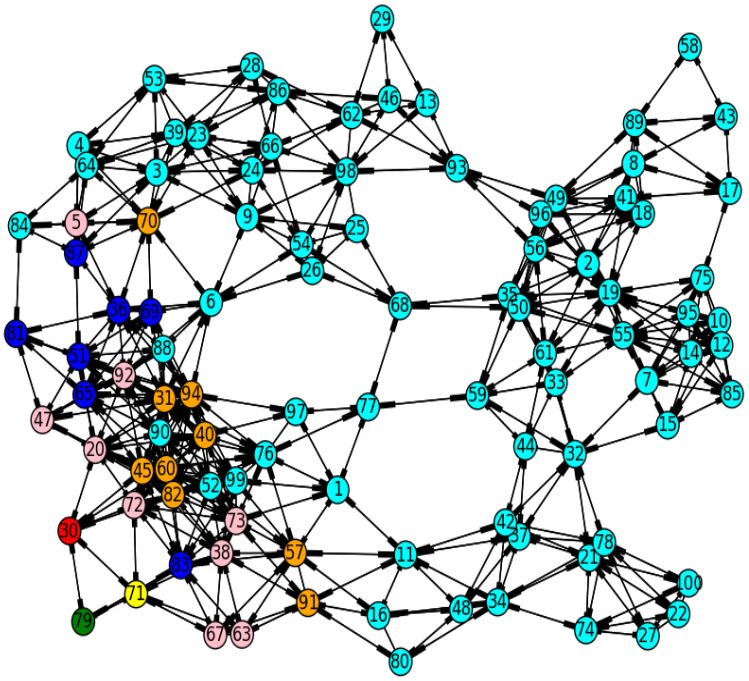


Figure 6.4: Relative Communication Graph Density

Figure 6.5 shows the top 30 ranked nodes based on the *Weighted Relative Communication Graph Density* metric. In 3-hop environments, the highest weighted relative communication graph density has nodes 68, 77 and 26.

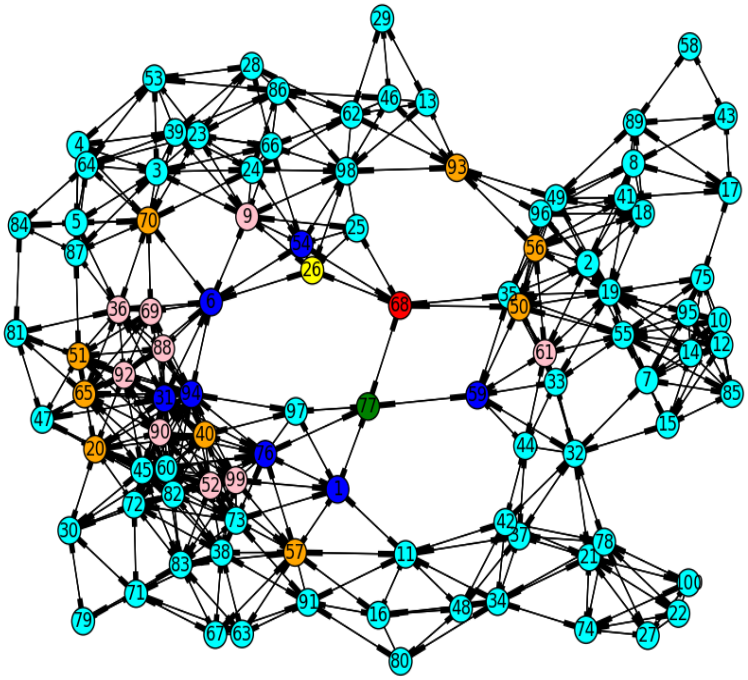


Figure 6.5: Weighted Relative Communication Graph Density

Clique-based metrics (degree of relieving)

Figures 6.6 - 6.8 show how we can use the clique based metrics to make ranking nodes. The task of all three metrics is to designate the degree of relieving nodes within their k-hop environment. Figure 6.6 shows node ranking created by $\mathcal{WCGR}_{sb}$. In the case of $\mathcal{WCGR}_{sb}$ only the size of cliques in the k-hop environment of the examined node is relevant. $\mathcal{WCGR}_{sb}$ marks primarily the nodes within the densest areas, just like $\mathcal{RCGD}$. Figure 6.7 shows node ranking created by $\mathcal{WCGR}_{vb}$. The significant difference between $\mathcal{WCGR}_{vb}$ and $\mathcal{WCGR}_{sb}$ is that $\mathcal{WCGR}_{vb}$ takes into consideration also the degree of neighbors. Figure 6.8 clearly shows that $\mathcal{CGR}_{vb}$ primarily focuses on centrally located nodes so the top 3 nodes and at least 80 % of the selected nodes are centrally located.

Figure 6.6 shows the top 30 ranked nodes based on the *Clique size-based Weighted Communication Graph Redundancy* metric. In 3-hop environments, the most relieved nodes has nodes 79, 30 and 81.

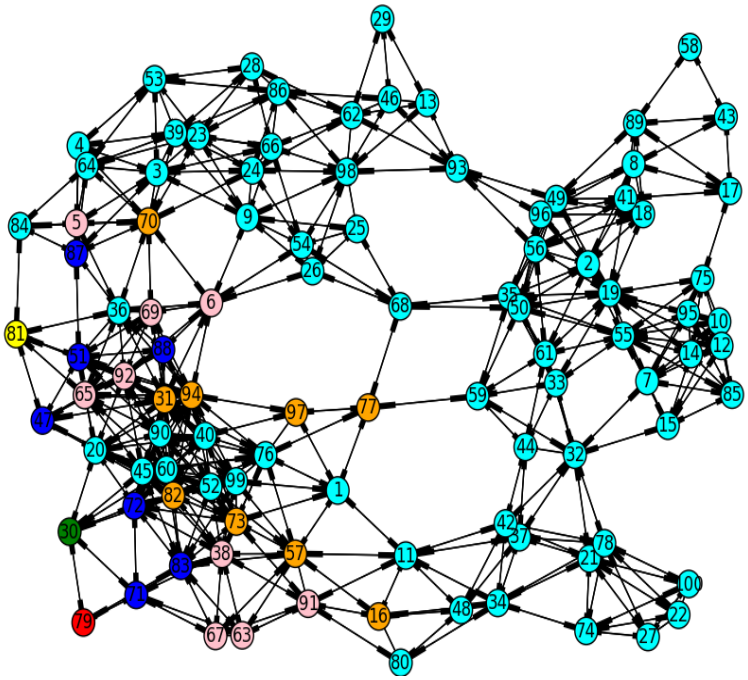


Figure 6.6: Clique size-based Weighted Communication Graph Redundancy

Figure 6.7 shows the top 30 ranked nodes based on the *Clique value-based Communication Graph Redundancy* metric. In 3-hop environments, the most relieved nodes has nodes 68, 77 and 26.

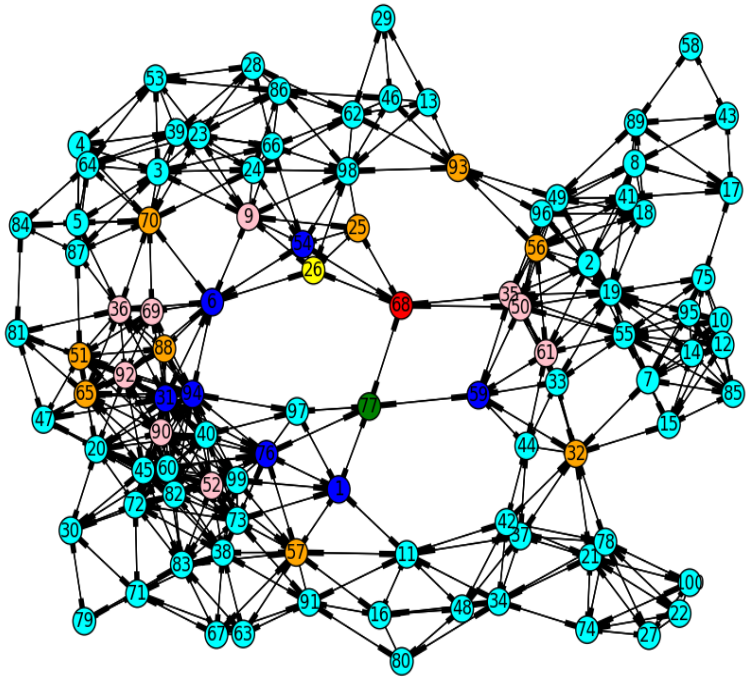


Figure 6.7: Clique value-based Communication Graph Redundancy

Figure 6.8 shows the top 30 ranked nodes based on the *Clique value-based Weighted Communication Graph Redundancy* metric. In 3-hop environments, the most relieved nodes has nodes 77, 68 and 26.

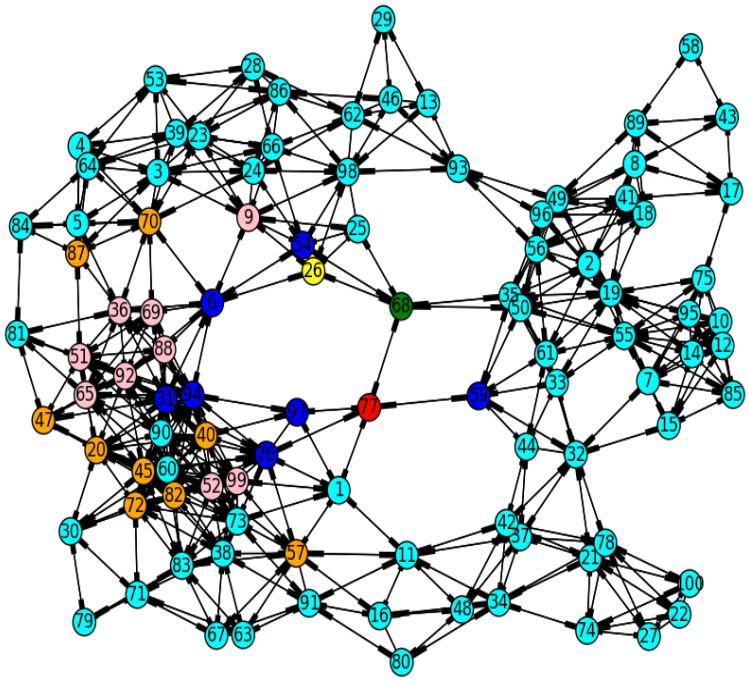


Figure 6.8: Clique value-based Weighted Communication Graph Redundancy

These figures show that $1-1$ densities and redundancy-based metrics similarly rank the

nodes. Why? The first reason is that the two concepts are closely related; if the density is high, then the redundancy is high as well. However, if the test is performed with directed graphs, there will be significant differences, because if node u can send a message to node v, then v may not be able to send a message to u, which means that the high density does not mean necessarily as well high redundancy.

6.6 Conclusions

In this chapter, we introduced several novel k-hop based density and redundancy metrics. We compared them to well-known graph metrics and we showed how they can be used for node ranking. Our primary goal was to define metrics that are able to rank nodes depending on their immediate environment within the whole network. Based on the results, we think that more sophisticated node ranking can be given using the new metrics. We primarily focused on modelling small heterogeneous networks. Metrics are defined so that they can be used also on networks where communication costs are different *(weighted directed graphs)*. Our further goal is to investigate also such networks. An interesting question is how to use these metrics to increase the efficiency of different *(Tx range-based, hierarchical)* topology control methods and how to use them in different hierarchical topological models *(e.g. clustering, cluster head selection)*.

7 Summary

In this dissertation, we have described new results for novel formal methods from modelling and analyzing *WSNs*. The results of this dissertation have been presented to the research community on international conferences and in journals.

In the introduction, we have summarized the major contributions and the impact of the work by the author. Chapter 2 gives an overview of the research area, the theoretical background and the related work.

In Part I, we have presented two zeroth-order logic-based models. We have introduced an algorithm, called $WnDGen$, and a propositional logic formula, called *Black-and-White 2 SAT problem*, to model a directed graph. Afterwards, we have presented two sequential *SAT* solvers (`CSFLOC`, `BaW 1.0`), and shown how to use them to solve *WSN*-based *SAT* problems. In the end of this part, we have shown how to develop techniques for using distributed computing resources to solve instances of the propositional satisfiability problem efficiently.

In Part III we, have introduced an *OMT* formalization of the optimization problem for *WSNs*, that provides a single-hop *WSN* simulation environment with one of the most common wireless sensor node types, propose several *OMT* benchmarks extracted from the *WSN* simulation. Also in this part, we have shown how to integrate this idea in search algorithms in the *OMT* framework and introduced a novel *OMT* solver called `Puli`.

Finally we have introduced several novel k-hop based density and redundancy metrics: ($\mathcal{WCGD}$, $\mathcal{RCGD}$, $\mathcal{WRCGD}$, $\mathcal{CGR}$, $\mathcal{WCGR}$ - *value and size-based*). We have compared them to known graph metrics and have shown that they can be used for node ranking.

At the beginning of the dissertation, three theses were formulated. The detailed results of the theses were described in sections 3.3 and 3.4 *(Thesis I)*, 5.3 and 5.4 *(Thesis II)* and 6.3, 6.4, 6.5 *(Thesis III)* respectively.

8 Összefoglalás

Ebben a disszertációban olyan új formális módszereket mutattunk be, amelyek alkalmasak vezeték nélküli szenzorhálózatok modellezésére és működésük elemzésére. Az értekezés eredményeit hazai és nemzetközi konferenciákon, valamint folyóiratokban publikáltuk.

A bevezetésben ismertettük a szerző által elért eredményeket, majd a második fejezetben áttekintést adtunk a kutatási területről, a téma elméleti hátteréről és a kapcsolódó munkákról.

Az I. fejezetben két nulladrendű logikai modellt mutattunk be. Egyrészt ismertettünk egy olyan algoritmust *(`WnDGen`)*, amellyel nehéz kombinatorikai benchmark-ok generálhatók, másrészt definiáltunk egy irányított gráfok reprezentálására alkalmas formulát *(Black-and- White 2-SAT)*.

Ezek után ismertetésre került két új szekvenciális *SAT* szolver *(`CSFLOC, BaW 1.0`)*, valamint megmutattuk, hogy ezek hogyan használhatók *WSN* alapú problémák megoldására. Ugyancsak ebben a fejezetben került bemutatásra, hogy hogyan oldhatók meg propozicionális kielégíthetőségi problémák elosztott számítási erőforrások használatával.

A harmadik fejezetben egy véletlenszerűen telepített és heterogén csomópontokból álló vezeték nélküli szenzorhálózat leírására alkalmas, rádió energia modellen alapuló, hálózat élettartamára optimalizálható *SMT* alapú reprezentációt ismertettünk. Definiáltunk egy új modell alapú inkrementális optimaliziálási eljárást. A fejezet végén bemutattunk egy probléma specifikus *OMT* solver-t, a `Puli`-t.

A negyedik fejezetben bemutatásra került három új sűrűség és három új redundancia alapú lokális metrika *($\mathcal{WCGD}$, $\mathcal{RCGD}$, $\mathcal{WRCGD}$, $\mathcal{CGR}$, $\mathcal{WCGR}$-klikk érték és méret alapú)*, melyeket összehasonlítottuk a szakirodalomból eddig ismert metrikákkal, és bemutattuk, hogyan használhatók csomópontok rangsorolására, továbbá klasszifikálásra.

A dolgozat elején három tézist fogalmaztunk meg. A tézisekhez köthető eredményeket rendre a 3.3 és a 3.4 *(I. tézis)*, az 5.3 és az 5.4 *(II. Tézis)* valamint a 6.3, a 6.4, és a 6.5 *(III. Tézis)* fejezetek voltak hivatottak bemutanti.

Bibliography

[1] S. Abdollahzadeh, N. J. Navimipour, *Deployment strategies in the wireless sensor network: A comprehensive review*, Computer Communications, Volumes 91–92, pp. 1-16, 2016.

[2] C. Aggarwal and K. Subbian, *Evolutionary Network Analysis: A Survey*, ACM Comput. Surv. 47, 1, Article 10, 36 pages, 2014.

[3] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, *Wireless sensor networks: a survey*, Computer Networks 38 , pp. 393 – 422, 2002.

[4] K. Akkaya, M. Younis, *A survey on routing protocols for wireless sensor networks*, Ad hoc networks, 3(3), 325-349, 2005.

[5] J. N. Al-Karaki, A. E. Kamal *Routing techniques in wireless sensor networks: a survey*, IEEE wireless communications, 11(6), 6-28, 2004.

[6] A. A. A. Alkhatib, G. S. Baicher, *Wireless sensor network architecture*, In International Conference on Computer networks and Communication Systems (ICNCS 2012) Vol. 35, pp. 11-15 , 2012.

[7] W. Anwar, M. Bakhtiari , A. Zainal , K. N. Qureshi, *Survey of Wireless Sensor Network Security and Routing Techniques*, Research Journal of Applied Sciences, Engineering and Technology Vol. 9(11), pp. 116-1026, 2015.

[8] B. Aspvall, M. F. Plass, R. E. Tarjan, *A Linear-Time Algorithm For Testing The Truth Of Certain Quantified Boolean Formulas*, Imformation Pprocessing Letters, pp. 121–123, 1979.

[9] R. Albert, A. L. Barabási, *Statistical mechanics of complex networks*, Reviews of Modern Physics, 2002, pp.74–77.

[10] D. P. Anderson, *BOINC: A System for Public-Resource Computing and Storage*, Proc. of GRID 2004, pp. 4–10, 2004.

[11] S. Andrei, *Counting for satisfiability by inverting resolution*, Artificial Intelligence Review, Volume 22, Issue 4, pp. 339–366, 2004.

[12] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, *Energy conservation in wireless sensor networks: A survey*, Ad Hoc Netw., vol. 7, no. 3, pp. 537–568, 2009.

[13] G. Audemard, L. Simon, *Glucose 2.3 in the SAT 2013 Competition*, In: Proceedings of SAT Competition 2013, pp. 42—43, 2013.

[14] A. Balint, A. Belov, D. Diepold, A. Gerber, M. Järvisalo, C. Sinz (eds), *Proceedings of SAT Challenge 2012: Solver and Benchmark Descriptions*, Department of Computer Science Series of Publications B, vol. B-2012-2, University of Helsinki, 2012.

[15] A. L. Barabási, R. Albert, H. Jeong, *Scale-free characteristics of random networks: the topology of the world-wide web*, Physica A: Statistical Mechanics and its Applications, pp. 69–77, 2000.

[16] C. Barrett, C. Tinelli *CVC3*, In Proc. of CAV 2007, volume 4590 of LNCS, Springer, pp. 298–302, 2007.

[17] H. Bennett and S. Sankaranarayanan, *Model Counting Using the Inclusion-Exclusion Principle*, Theory and Applications of Satisfiability Testing - SAT 2011 Lecture Notes in Computer Science, Volume 6695, pp. 362–363, 2011.

[18] Z. Benenson, Zinaida , F. C. Freiling, P. M. Cholewinski, *Advanced Evasive Data Storage in Sensor Networks*, Proc. Int. Conf. on Mobile Data Management, pp. 146–151, 2007.

[19] A. Bharathidasan, V. A. S. Ponduru, *Sensor Networks: An Overview*, Department of Computer Science, University of California, Davis, CA, 2002.

[20] M. Bhardwaj, A. Chandrakasan, *Bounding the Lifetime of Sensor Networks Via Optimal Role Assignments*, INFOCOM 2002, pp. 1587-1596, 2002.

[21] A. Biere, A. Cimatti, E. Clarke, Y. Zhu, (1999, March). *Symbolic model checking without BDDs. In International conference on tools and algorithms for the construction and analysis of systems*, Springer, Berlin, Heidelberg, pp. 193-207, 1999.

[22] A. Biere, M. Heule, H. van Maaren, T. Walsh, *Handbook of Satisfiability*, IOS Press, Amsterdam, 2009.

[23] A. Biere, *Lingeling, Plingeling, PicoSAT and PrecoSAT at SAT Race 2010*, Technical Report 10/1, FMV Reports Series, JKU, 2010.

[24] A. Biere, *Lingeling and Friends Entering the SAT Challenge 2012*, Haifa Verification Conference, Department of Computer Science Series of Publications B, vol. B-2012-2, pp. 33–34, 2012.

[25] E. Birnbaum and E. L. Lozinskii, *The good old Davis-Putnam procedure helps counting models*, Journal of Artificial Intelligence Research, Volume 10, 457–477, 1999.

[26] N. Bjørner, A. Phan, L. Fleckenstein, *μZ - An Optimizing SMT*, Proc. Int. Conf. for Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS Vol. 9206, pp. 194–199, 2015.

[27] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D. U. Hwang, *Complex networks: Structure and dynamics*, Physics Report, 2006.

[28] M.Bolic, I. Stojmenovic (eds.) *RFID Systems - Research Trends and Challenges*, John Wiley & Sons 576 Pages, 2010.

[29] P. Bonacich, *A Technique for Analyzing Overlapping Memberships*, Sociological Methodology, San Francisco: Jossey-Bass, pp. 176-85, 1972.

[30] P. Bonacich *Factoring and Weighting Approaches to Status Scores and Clique Identification*, Journal of Mathematical Sociology pp. 113-20, 1972.

[31] J.A. Bondy, S. Uppaluri , M. Ramachandra, *Graph theory with applications*, Vol. 290. London: Macmillan, 1976.

[32] D. Braginsky, D. Estrin, *Rumor Routing Algorithm for Sensor Networks*, Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (WSNA); Atlanta, GA, USA. September, pp. 22–31, 2002.

[33] T. Braun, A. Kassler, M. Kihl, V. Rakocevic, V. Siris, G. Heijenk, *Multihop Wireless Networks*, Traffic and QoS Management in Wireless Multimedia Networks pp. 201-265, 2009.

[34] R. Brummayer, A. Biere, *Boolector: An efficient SMT solver for bit-vectors and arrays* In International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Springer, Berlin, Heidelberg, pp. 174–177, 2009.

[35] R. Bruttomesso, A. Cimatti, A. Franzén, A. Griggio, R. Sebastiani, *The mathsat 4 smt solver*, In International Conference on Computer Aided Verification Springer, Berlin, Heidelberg , pp. 299-303, 2008.

[36] R. E. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Trans. Comput., pp. 677–691, 1986.

[37] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang, *Symbolic model checking: 1020 states and beyond*, Information and computation, 98(2), pp. 142-170, 1992.

[38] M.Cardei, M. and D. Ding-Zhu, *Improving Wireless Sensor Network Lifetime through Power Aware Organization*, Wireless Networks, Vol 11. N. 3. pp. 333–340, 2005.

[39] M.Cardei,M. Thai, Y. Li, W. Wu, *Energy-efficient target coverage in wireless sensor networks*, INFOCOM 2005. 24th annual joint conference of the IEEE computer and communications societies. Proceedings IEEE, Vol. 3, pp. 1976–84, 2005.

[40] M.Cardei, *Coverage Problems in Sensor Networks*, Handbook of Combinatorial Optimization, pp. 899-927, 2013.

[41] B. Carter R. K. Ragade *A probabilistic model for the deployment of sensors*, Conference: Sensors Applications Symposium, 2009.

[42] W. P. Chen, J. Hou, L. Sha, *Dynamic clustering for acoustic target tracking in wireless sensor networks*, IEEE Trans. Mob. Comput. 3, 258–271, 2004.

[43] X. Cheng, D-Z. Du, L. Wang, B. Xu, *Relay sensor placement in wireless sensor networks*, Wireless Networks Vol.14. ,pp. 347–55, 2008.

[44] W. Chrabakh, R. Wolski, *GridSAT: A Chaff-based Distributed SAT Solver for the Grid*, Proc. of SC'03, pp. 37–49, 2003.

[45] W. Chrabakh, R. Wolski, *GrADSAT: A Parallel SAT Solver for the Grid.*, Technical report, UCSB Computer Science, 2003.

[46] E. M. Clarke, J. M. Wing, *Formal methods: State of the art and future directions*, ACM Computing Surveys (CSUR), 28(4), pp. 626-643, 1996.

[47] O. Chanseok, *Improving SAT Solvers by Exploiting Empirical Characteristics of CDCL, PhD thesis*, New York University, 2016.

[48] T. F. Coleman, J. Moré, *Estimation of Sparse Jacobian Matrices and Graph Coloring Blems* , SIAM Journal on Numerical Analysis, pp.187–209, 1983.

[49] S. A. Cook, *The Complexity of Theorem-Proving Procedures*, Proc of STOC'71, pp. 151–158, 1971.

[50] M. Davis, G. Logemann, D. Loveland, *A Machine Program for Theorem Proving*, Commun. ACM, vol. 5, no. 7, pp. 394–397, 1962.

[51] E. W. Dijkstra, *A Discipline of Programming*, NJ: Prentice Hall, Ch. 25. 1976.

[52] E. W. Dijkstra, *Finding the Maximum Strong Components in a Directed Graph*, In Selected Writings on Computing: A personal Perspective, Texts and Monographs in Computer Science, Springer New York, pp. 22–30, 1982.

[53] O. Dubois, *Counting the Number of Solutions for Instances of Satisfiability*, Theoretical Computer Science, Volume 81, pp. 49-–64, 1991.

[54] B. Dutertre and L. de Moura, *The Yices SMT solver* Tool paper at http://yices.csl.sri.com/toolpaper.pdf, 2006

[55] H. Echoukairi, K. Bourgba, M. Ouzzif, *A Survey on Flat Routing Protocols in Wireless Sensor Networks*, In Advances in Ubiquitous Networking, pp. 311-324, 2015.

[56] H. B. Enderton *A Mathematical Introduction to Logic* Undergraduate Texts in Mathematics. Academic Press, second edition edition, 2000.

[57] Y. Yu, D. Estrin, R. Govindan, *Geographical and energyaware routing: a recursive data dissemination protocol for wireless sensor networks*, UCLA Computer Science Department Technical Report, UCLA-CSD TR-01-0023, May 2001.

[58] S. Even, R.E. Tarjan, *Network flow and testing graph connectivity*, SIAM journal on computing, 4(4), pp. 507-518, 1975.

[59] M. Fitting, *First-order logic and automated theorem proving*, Springer Science and Business Media, 2012.

[60] C. Fischione, P. Park, P. Di Marco, K. H. Johansson, *Chapter 9 Design Principles of Wireless Sensor Networks Protocols for Control Applications* , 2011.

[61] M. Fourman, P. Palmer, R. M. Zimmer, *Proof and synthesis*, In Computer Design: VLSI in Computers and Processors, 1988. ICCD'88., Proceedings of the 1988 IEEE International Conference on pp. 600-603, 1988.

[62] L. C. Freeman, *set of measures of centrality based on betweenness*, Sociometry, pp. 35-41, 1977.

[63] L. C. Freeman, *Centrality in social networks conceptual clarification*, Social Networks, Vol. 1, Issue 3, pp.205, 1978.

[64] L. C. Freeman, D. Roeder, R. R. Mulholland, *Centrality in social networks: II. Experimental results*, Social networks, 2(2), 119-141.(1979)

[65] J.W. Freeman, *Improvements to Propositional Satisfiability Search Algorithms. Ph.D. Dissertation*, Department of Computer and Information Science, University of Pennsylvania, 1995.

[66] H. Nagamochi, T. Ibaraki, *Algorithmic Aspects of Graph Connectivity*, Cambridge University Press, New York, NY, USA. 2008.

[67] Y. Hamadi S. Jabbour, L. Sais, *ManySAT: a Parallel SAT Solver*, Journal on Satisfiability, Boolean Modeling and Computation, vol. 6, pp. 245–262, 2009.

[68] Y. Hamadi, S. Jabbour, C. Piette, L. Saïs, *Deterministic Parallel DPLL*, JSAT, vol. 7. no. 4, pp. 127–132, 2011.

[69] Y. Xu, J. Heidemann, D. Estrin, *Geography-informed energy conservation for ad hoc routing*, in: Proceedings of the 7th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom01), Rome, Italy, July 2001.

[70] J. M. Hernandez , P. Van Mieghem, *Classification of graph metrics*, Delft University of Technology, Technical Report 2628 CD Delft, 2011.

[71] M. Heule, *March: Towards a Look-ahead SAT Solver for General Purposes*, Master thesis, TU Delft, The Netherlands, 2004.

[72] M. Heule, O. Kullmann, S. Wieringa, A. Biere, *Cube and Conquer: Guiding CDCL SAT Solvers by Lookaheads*, Lecture Notes in Computer Science, vol. 7261, pp. 50–65, 2011.

[73] A. Hossain ; P. K. Biswas , S. Chakrabarti, *Sensing Models and Its Impact on Network Coverage in Wireless Sensor Network*, 2008 IEEE Region 10 and the Third international Conference on Industrial and Information Systems, 2008.

[74] A. E. J. Hyvärinen, T. Junttila, I. Niemelä, *Partitioning SAT instances for distributed solving*, Proc. of LPAR'10, pp. 372–386, 2010.

[75] H. N. Gabow, *Path-based depth-first search for strong and biconnected components*, Information Processing Letters, Volume 74, Issues 3–4, pp: 107-114, 2000.

[76] A. V. Gelder, *Propositional Search with k-Clause Introduction Can be Polynomially Simulated by Resolution*, Proceedings of the 5th International Symposium on Artificial Intelligence and Mathematics, 1998.

[77] H. Geuvers, *Proof assistants: History, ideas and future*, Sadhana, 34(1), 3-25, 2009.

[78] C.P. Gomes, B. Selman, H. Kautz, *Boosting combinatorial search through randomization*, National Conference on Artificial Intelligence, pp. 431-–437, 1998.

[79] C. P. Gomes, A. Sabharwal, B. Selman, *Model Counting*, Chapter 20 of Handbook of Satisfiability, IOS Press, Amsterdam, 2009.

[80] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, *Internet of Things (IoT): A vision, architectural elements, and future directions*, Future Generat. Comput. Syst., vol. 29, no. 7, pp. 1645–1660, 2013.

[81] L. Hellerman, *A Catalog of Three-Variable Or-Invert and And-Invert Logical Circuits*, IEEE Transactions on Electronic Computers, pp. 198–223, 1963.

[82] S. Hussain, O. Islam, *An Energy Efficient Spanning Tree Based Multi-Hop Routing in Wireless Sensor Networks*, Proceedings of the Wireless Communications and Networking Conference, Hong Kong, 11–15, pp. 4383–4388, 2007.

[83] C. Intanagonwiwat, R. Govindan, D. Estrin, *Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Networks.*, Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking (MOBICOM); Boston, MA, USA. August, 2000.

[84] K. Iwama, *CNF-satisfiability test by counting and polynomial average time*, SIAM Journal on Computing, Volume 18, Issue 2, pp. 385–391, 1989.

[85] M. Jacobsson and C. Orfanidis, *Using software-defined networking principles for wireless sensor networks*, in Proc. 11th Swedish Nat. Comput. Netw. Workshop, Karlstad, Sweden, 2015.

[86] T. Jebelean and G. Kusper, *Multi–Domain Logic and its Applications to SAT*, SYNASC 2008, IEEE Computer Society Press, pp. 3–8, 2008.

[87] J. Johannsen, *The Complexity of Pure Literal Elimination*, In: Giunchiglia E., Walsh T. (eds) SAT 2005. Springer, Dordrecht 2005.

[88] C. B. Jones, *Systematic software development using VDM*, Vol. 2, Englewood Cliffs: Prentice Hall, 1990.

[89] J. J. Joyce, *Formal verification and implementation of a microprocessor In VLSI Specification*, Verification and Synthesis pp. 129-157 1988.

[90] P. Kacsuk, J. Kovács, Z. Farkas, A. C. Marosi, G. Gombás, Z. Balaton, *SZTAKI Desktop Grid (SZDG): A Flexible and Scalable Desktop Grid System*, Journal of Grid Computing, vol. 7, no. 4, pp. 439–461, 2009.

[91] K. W. Kai-Wei Fan, S. Liu, P. Sinha, *Structure-free data aggregation in sensor networks*, IEEE Trans. Mob. Comput. 6, 929–942, 2007.

[92] R. M. Karp, *Reducibility among Combinatorial Problems*, Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, pp. 20–22, 1972.

[93] N. Kaur, M. S. Kahlon, *A Review on Reactive and Proactive Wireless Sensor Networks Protocols*, International Journal of Computer Applications, Vol. 95. No. 11, 2014.

[94] H. Katebi, K.A. Sakallah, J. P. Marques-Silva, *Empirical Study of the Anatomy of Modern Sat Solvers*, In: Sakallah K.A., Simon L. (eds) Theory and Applications of Satisfiability Testing - SAT 2011. SAT 2011. Lecture Notes in Computer Science, vol 6695. Springer, Berlin, Heidelberg 2011.

[95] C. Kern, M. R. Greenstreet, *Formal verification in hardware design: a survey*, ACM Transactions on Design Automation of Electronic Systems (TODAES), 4(2), pp. 123-193, 1999.

[96] L. Kong, Q. PangYang, *The Comparison Study of Flat Routing and Hierarchical Routing in Ad Hoc Wireless Networks*, Networks, ICON '06. 14th IEEE International Conference on Volume: 1, 2006.

[97] , W. Kong, M. and Hang Li, A. Long and Fukuda, *An SMT-based Accurate Algorithm for the K-Coverage Problem in Sensor Network*, Proc. 8th Int. Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM), pp. 240–245, 2014.

[98] L. Kovács, A. Voronkov *First-order theorem proving and Vampire*, In International Conference on Computer Aided Verification, Springer, Berlin, Heidelberg, pp. 1–35, 2013.

[99] J. Kulik, W. Heinzelman, H. Balakrishnan , *Negotiation-based Protocols for Disseminating Information in Wireless Sensor Networks* Wirel. Netw. Vol 8, pp. 169–185. 2002.

[100] O. Kullmann, *New methods for 3-SAT decision and worst-case analysis*, Theoretical Computer Science, 223(1-2): pp. 1–72, 1999.

[101] O. Kullmann, *On a generalization of extended resolution*, Discrete Applied Mathematics, 96-97(1-3) pp. 149–176, 1999.

[102] O. Kullmann, *Investigating the Behaviour of a SAT Solver on Random Formulas*, Technical Report CSR 23-2002, Swansea University, Computer Science Report Series, 2002.

[103] G. Kusper, *Solving and Simplifying the Propositional Satisfiability Problem by Sub-Model Propagation*, PhD thesis, Johannes Kepler University Linz, RISC Institute, pp. 1–146, 2005.

[104] G. Kusper, *Finding Models for Blocked 3-SAT Problems in Linear Time by Systematical Refinement of a Sub-Model*, Lecture Notes in Computer Science 4314, pp. 128–142, 2007.

[105] R. P. Langlands, *Problems in the theory of automorphic forms to Salomon Bochner in gratitude*, Lectures in Modern Analysis and Applications III, pp. 18–61, 1970.

[106] M. T. Lazarescu *Design of a WSN platform for long-term environmental monitoring for IoT applications*, IEEE Journal on emerging and selected topics in circuits and systems, 3(1), pp. 45-54, 2013.

[107] C.M. Li and Anbulagan, *Look-Ahead versus Look-Back for Satisfiability Problems*, Lecture Notes in Computer Science, vol. 1330, pp. 342–356, 1997.

[108] X. Leroy, *Formal certification of a compiler back-end or: programming a compiler with a proof assistant*, In ACM SIGPLAN Notices, Vol. 41, No. 1, pp. 42-54, 2006.

[109] X. Leroy, *Formal verification of a realistic compiler*, Communications of the ACM, 52(7), pp. 107-115, 2009.

[110] A. Lesne, *Complex Networks: from Graph Theory to Biology* Letters in Mathematical Physics, pp.235–262, 2006.

[111] M. Li, Z. Li, A. V. Vasilakos, *A Survey on Topology Control in Wireless Sensor Networks: Taxonomy, Comparative Study, and Open Issues*, in Proceedings of the IEEE, vol. 101, no. 12, pp. 2538-2557, 2013.

[112] S. Li and X. Wang, *Wireless Sensor Hierarchical Networks*, Chapter 5 in Handbook of Sensor Networking, CRC Press, 2015.

[113] Y. Li , C. Vu , C. Ai , G. Chen , Y. Zhao, *Transforming Complete Coverage Algorithms to Partial Coverage Algorithms for Wireless Sensor Networks*, IEEE Transactions on Parallel and Distributed Systems Vol. 22, pp. 695-703, 2011.

[114] Y. Li, A. Albarghouthi, Z. Kincaid, A. Zachary, M. Chechik, *Symbolic Optimization with SMT Solvers*, Proc. ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), pp. 607–618, 2014.

[115] X. Liu, H. Zhou, S. Xiong, K. M. Hou, C. De Vaulx, H. Shi, *Development of a Resource-Efficient and Fault-Tolerant Wireless Sensor Network System*, In Dependable Computing and Internet of Things (DCIT), 2015 2nd International Symposium on IEEE, pp. 122-127, 2015.

[116] J. H. Liang, V. Ganesh, P. Poupart, K. Czarnecki, *Learning Rate Based Branching Heuristic for SAT Solvers*, 19th International Conference on Theory and Applications of Satisfiability Testing SAT 2016, 2016.

[117] J. H. Liang, V. Ganesh, P. Poupart, K. Czarnecki, *Exponential Recency Weighted Average Branching Heuristic for SAT Solvers*, In: Proceedings of AAAI-16, 2016.

[118] J. H. Liang, Chanseok Oh, M. Mathews, C. Thomas, C. Li, V. Ganesh, *Machine Learning-based Restart Policy for CDCL SAT Solvers*, 21st International Conference on Theory and Applications of Satisfiability Testing (SAT 2018), 2018.

[119] S. Lindsey, C.S. Raghavendra, *PEGASIS: Power-Efficient Gathering in Sensor Information Systems*, Proceedings of the Aerospace Conference; Big Sky, MT., pp. 1125–1130, 2002.

[120] B. Liu B, P. Brass, O. Dousse, P. Nain, D. Towsley, *Mobility improves coverage of sensor networks*, Proceedings of the 6th ACM international symposium on mobile ad hoc networking and computing, ser. MobiHoc'05. New York, NY, USA: ACM, pp. 300–8, 2005.

[121] E. L. Lozinskii, *Counting propositional models*, Information Processing Letters, Volume 41, pp. 327–332, 1992.

[122] R. Luce, A. Duncan, D. Albert , *A method of matrix analysis of group structure*, Psychometrika, pp. 95–116, 1949.

[123] H. Luo, G.J. Pottie, *Designing routes for source coding with explicit side information in sensor networks*, IEEE Trans. Netw., 15 (6), pp. 1401-1413, 2007.

[124] Y. S. Mahajan, Z. Fu, S. Malik, *Zchaff2004: An Efficient SAT Solver*, Lecture Notes in Computer Science: Theory and Applications of Satisfiability Testing, vol. 3542, pp. 360–375, 2005.

[125] Q. Mamun *A Qualitative Comparison of Different Logical Topologies for Wireless Sensor Networks*, Sensors, 12(11), pp. 14887-14913, 2012.

[126] E. Manjeshwar, D.P. Agrawal *TEEN: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks*, Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS), San Francisco, CA, USA. pp. 2009–2015, 2001.

[127] F. Mansourkiaie, M.H. Ahmed, *Optimal and near-optimal cooperative routing and power allocation for collision minimization in wireless sensor networks*, IEEE Sens. J., 16 (5), pp. 1398-1411, 2016.

[128] J. Marques-Silva, K. A. Sakallah, *GRASP: A new search algorithm for satisfiability*, Srivas, M., Camilleri, A. (eds.) FMCAD 1996. LNCS, vol. 1166, pp. 220—227, 1996.

[129] J. Marques-Silva, K. A. Sakallah, *GRASP-A search algorithm for propositional satisfiability*, IEEE Transactions on Computers 48(5), pp. 506-–521, 1999.

[130] S. Megerian, F. Koushanfar F, G. Qu, G. Veltri, M. Potkonjak *Exposure in wireless sensor networks: theory and practical solutions*, Wireless Networks Vol. 8. pp. 443-454 2002.

[131] S. Minato, *Zero-suppressed BDDs for Set Manipulation in Combinatorial Problems*, Proceedings of the 30th International Design Automation Conference, pp.272–277, 1993.

[132] M. Mo, D. Qiao and Z. Wang, *Mostly-sleeping wireless sensor networks: Connectivity, k-coverage, and -lifetime*, In Proceedings of the 43rd Annual Allerton Conference on Communication, Control, and Computing, pp. 1-10, 2005.

[133] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, S. Malik, *Engineering an efficient SAT solver*, Design Automation Conference, pp. 530-–535, 2001.

[134] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, *Chaff: Engineering an Efficient SAT Solver*, Proc. of DAC'01, pp. 530–535, 2001.

[135] L. M. de Moura and N. Bjørner *Z3: An efficient SMT solver*, In Proc. of TACAS 2008, volume 4963 of LNCS, pp. 337–340, 2008.

[136] I. Munro, *Efficient Determination of the Transitive Closure of a Directed Graph*, Information Processing Letters, 1(2), pp: 56–58, 1971.

[137] B. Nagy, *The Languages of SAT and n-SAT over Finitely Many Variables are Regular*, Bulletin of the EATCS, no. 82, pp. 286–297, 2004.

[138] B. Nagy, *On the Notion of Parallelism in Artificial and Computational Intelligence*, Proc. of HUCI 2006, pp. 533–541, 2006.

[139] B. Nagy, *On Efficient Algorithms for SAT*m Proc. of CMC 2012, Lecture Notes in Computer Science, vol. 7762, pp. 295–310, 2013.

[140] C.Narmadha, P.Marichamy, A.E.Narayanan, *A Survey on Hierarchical-Based Routing Protocols for Wireless Sensor Networks, International Journal of Pure and Applied Mathematics* Vol. 119 No. 16, 3663-3676, 2018.

[141] *G. Nelson, D. C. Oppen*, *Simplification by cooperating decision procedures*, ACM Transactions on Programming Languages and Systems (TOPLAS), 1(2), 245-257, 1979.

[142] J. Nešetřil, P. Ossona de Mendez, *First order properties on nowhere dense structures* , Journal of Symbolic Logic, pp. 868–887, 2010.

[143] M. E. J. Newman, *Networks An Introduction*, Oxford University Press, 2010.

[144] A. Nordrum, *Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated*, https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated IEEE, 2016.

[145] A. Norouzi, A. H. Zaim, *An Integrative Comparison of Energy Efficient Routing Protocols in Wireless Sensor Network*, Wireless Sensor Network, Vol. 4. pp. 65-75, 2012.

[146] L. C. Paulson, *Isabelle: A generic theorem prover*, Vol. 828, Springer Science and Business Media, 1994.

[147] M. Posypkin, A. Semenov, O. Zaikin, *Using BOINC Desktop Grid to Solve Large Scale SAT Problems*, Computer Science, vol. 13, no. 1, pp. 25–34, 2012.

[148] V. Potdar, A. Sharif, and E. Chang *Wireless sensor networks: A survey* in Proc. Int. Conf. Adv. Inf. Netw. Appl. Workshops, pp. 636–641, 2009.

[149] A. Puggelli, M.M.R. Mozumdar, L. Lavagno, A.L.S. Vincentelli, *Routing-aware design of indoor wireless sensor networks using an interactive tool*, IEEE Syst. J., 9 (3), pp. 714-727, 2015.

[150] P. Purdom, *A transitive closure algorithm*, BIT Numerical Mathematics 10.1, pp. 76-94, 1970.

[151] J. Qadir and O. Hasan *Applying formal methods to networking: Theory, techniques, and applications*, IEEE Commun. Surveys Tuts., vol. 17, no. 1, 1st Quart., pp. 256–291, 2015.

[152] , D. Qi, A.H. Saeed, A.S., *Provable configuration planning for wireless sensor networks*, Proc. 8th Int. Conf. on Network and Service Management (CNSM) and Workshop on Systems Virtualization Management (SVM), pp. 316–321, 2012.

[153] W. Rabiner, J. Kulik, H. Balakrishnan *Adaptive Protocols for Information Dissemination in Wireless Sensor Networks*, Proceedings of the Fifth Annual International Conference on Mobile Computing and Networking (MOBICOM), Seattle, WA, USA. , pp. 174–185, 1999.

[154] R. Rajaraman, *Topology control and routing in ad hoc networks: A survey*, ACM SIGACT News, 33(2), pp. 60-73, 2002.

[155] R. S. Rastko, V. , A. S. Phoha, *WSN Architecture*, Wireless Sensor Networks, pp. 37-81, 2016.

[156] D.N. Rewadkar and M.P. Madhukar, *An Adaptive Routing Algorithm Using Dynamic TTL for Data Aggregation in Wireless Sensor Network*, 2nd International Conference on Current Trends in Engineering and Technology ICCTET 14 , pp. 192–197, 2014.

[157] V. Rodoplu, T.H. Ming, *Minimum energy mobile wireless networks*, IEEE Journal of Selected Areas in Communications 17 (8) 1333–1344, 1999.

[158] B. Roy *Transitivité et connexité*, C R Acad Sci Paris 249, pp: 216–218, 1959.

[159] J. O'Rourke, *Art gallery theorems and algorithms*, New York, NY, USA: Oxford University Press, Inc.; 1987.

[160] P. Santi, *Topology Control in Wireless Ad Hoc and Sensor Networks* , ACM Computing Surveys, pp. 164–194, 2005.

[161] P. Sausen, M. Spohny, A. Perkusichy, *Energy Efficient Blind Flooding in Wireless Sensors Networks*, Proceedings of the 34th Annual Conference of IEEE Industrial Electronics, Orlando, FL, USA, 10–13, pp. 1736–1741, 2008.

[162] A. Sarkar, T. S. Murugan, *Routing protocols for wireless sensor networks: What the literature says?*, Alexandria Engineering Journal, Volume 55, Issue 4, pp. 3173-3183, 2016.

[163] J. Scott, *Social network analysis*, Sage, 2017.

[164] R. Sebastiani, P. Trentin, *Pushing the Envelope of Optimization Modulo Theories with Linear-Arithmetic Cost Functions*, Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS Vol. 9035, pp. 335-349, 2015.

[165] R. R. Selmic, V. V. Phoha, A. Serwadda, *Topology, Routing, and Modeling Tools*, In Wireless Sensor Networks pp. 23-36, 2016.

[166] M. Sharir, *A strong-connectivity algorithm and its applications in data flow analysis*, Computers And Mathematics with Applications, pp. 67–72, 1981.

[167] R. E. Shostak, *A practical decision procedure for arithmetic with function symbols*, Journal of the ACM (JACM), 26(2), 351-360, 1979.

[168] K. Simon, *An improved algorithm for transitive closure on acyclic digraphs*, Theor Comput Sci 58, pp. 325–346, 1988.

[169] M. Srbinovska, C. Gavrovski, V. Dimcev, A. Krkoleva, V. Borozan, *Environmental parameters monitoring in precision agriculture using wireless sensor networks*, Journal of Cleaner Production, 88, pp. 297-307, 2015.

[170] R. Sebastiani and P. Trentin, *OptiMathSAT: A Tool for Optimization Modulo Theories*, Proc. Int. Conf. on Computer-Aided Verification (CAV),LNCS Vol. 9206, pp. 447–454, 2015.

[171] I. Streinu, L. Theran, *Sparse hypergraphs and pebble game algorithms*, European Journal of Combinatorics, pp. 1944–1964, 2009.

[172] Y. Tanaka, *A Dual Algorithm for the Satisfiability Problem*, Information Processing Letters, Volume 37, pp. 85—89, 1991.

[173] D. Tang, T. Li, J. Ren, J. Wu, *Cost-Aware SEcure Routing (CASER) protocol design for wireless sensor networks*, IEEE Trans. Parallel Distrib. Syst., 26 (4), pp. 960-973, 2015.

[174] S. Taneja, A. Kush,, *A survey of routing protocols in mobile ad hoc networks*, International Journal of innovation, Management and technology, 1(3), 279. 2010.

[175] R. Tarjan,, *Depth-First Search and Linear Graph Algorithms*, SIAM Journal on Computing, Vol. 1, No. 2, pp. 146-160, 1972.

[176] D. Tian, G. D. Georganas, *A Coverage-preserving Node Scheduling Scheme for Large Wireless Sensor Networks*, Proc. Int. Workshop on Wireless Sensor Networks and Applications, WSNA'02, pp. 32–41, 2002.

[177] A. K. Tripathy, S. Chinara, *Evolution of Virtual Clustering in Wireless Sensor Networks*, Wireless Sensor Networks to theory to application- chapter 15, CRC Press, 2015.

[178] G. Tseitin, *On the Complexity of Derivation in Propositional Calculus*, Seminars in Mathematics Volume 8: Studies in Constructive Mathematics and Mathematical Logic, Part II: pp. 115–125, 1968.

[179] M. T. Van Genuchten, *A closed-form equation for predicting the hydraulic conductivity of unsaturated soils*, I. Soil science society of America journal, 44(5), pp. 892-898, 1980.

[180] H. Wang, D. Peng, W. Wang, H. Sharif, H. H. Chen, *Cross-layer routing optimization in multirate wireless sensor networks for distributed source coding based applications*, IEEE Trans. Wireless Commun., 7 (10) , pp. 3999-4009, 2008.

[181] Y. Wang, *Topology Control for Wireless Sensor Networks* , Springer - Wireless Sensor Networks and Applications, pp. 113–147, 2008.

[182] Y. C. Wang, C. C. Hu, Y. C. Tseng, *Efficient Placement and Dispatch of Sensors in a Wireless Sensor Network*, IEEE Transactions on Mobile Computing, Vol. 7, pp. 262–274, 2008.

[183] S. Warshall, *A theorem on boolean matrices*, J Assoc Comput Mach pp. 11–12, 1962.

[184] S. Wasserman, K. Faust, *Social network analysis: Methods and applications* , Cambridge university press, v. 4. 1994.

[185] D. J. Watts, S. H. Strogatz, *Collective dynamics of 'small-world' networks*, Nature, pp. 440–442, 1998.

[186] A. Weil, *Uber die Bestimmung Dirichletscher Reihen durch Funktionalgleichungen*, Mathematische Annalen, 149–156, 1967.

[187] A. J. Wiles, *Modular elliptic curves and Fermat's Last Theorem*, ANNALS OF MATH, pp. 443–551, 1995.

[188] X. Xu, X. Y. Li, X. Mao, S. Tang, S. Wang, *A delay-efficient algorithm for data aggregation in multihop wireless sensor networks.*, IEEE Trans. Parall. Distrib. Syst. 22, 163–175, 2011.

[189] Y. Yao, J. Gehrke, *The Cougar Approach to In-Network Query Processing in Sensor Networks*, SIGMOD Rec. , Vol. 31. 9–18, 2002.

[190] M. Ye, C. Li, G. Chen, J. Wu, *EECS: An Energy Efficient Clustering Scheme in Wireless Sensor Networks*, Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference, Phoenix, AZ, USA, pp. 535–540, 2005.

[191] J. Yick, B. Mukherjee, and D. Ghosal, *Wireless sensor network survey*, Comput. Netw., vol. 52, no. 12, pp. 2292–2330, 2008.

[192] O. Younis, S. Fahmy, *"HEED: a hybrid, energy-efficient, distributed clustering approach for ad hoc sensor networks*, IEEE Transactions on Mobile Computing, Vol. 3, No. 4, pp. 366-379, 2004.

[193] H. Zhang, M. E. Stickel, *An Efficient Algorithm for Unit Propagation*, In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI-MATH'96), pp. 166–169, 1996.

[194] H. Zhang, J. Hou, *On deriving the upper bound of -lifetime for large sensor networks*, In MobiHoc '04: Proceedings of the 5th ACM international symposium on Mobile ad hoc networking and computing, pp. 121–132, 2004.

[195] J. Zhang, T. Yan, S. Son, *Deployment strategies for differentiated detection in wireless sensor networks*, Sensor and Ad Hoc Communications and Networks Vol. 1, pp. 316–325, 2006.

[196] X. M. Zhang, Y. Zhang, Y., F. Yan, A. V. Vasilakos *Interference-based topology control algorithm for delay-constrained mobile ad hoc networks*, IEEE Transactions on Mobile Computing, 14(4), pp. 742-754, 2015.

[197] C. Zhu, C. Zheng, L. Shu, G. Han *A survey on coverage and connectivity issues in wireless sensor networks*, Journal of Network and Computer Applications Vol. 35, pp. 619–632, 2012.

Author Bibliography

Journal Papers

[198] **Cs. Biró**, G. Kusper *Equivalence of Strongly Connected Graphs and Black-and-White 2-SAT Problems*, Miskolc Mathematical Notes, Vol. 19, No. 2, pp. 755-768, 2018.

[199] **Cs. Biró**, G. Kusper *Some k-hop Based Graph Metrics and Node Ranking in Wireless Sensor Networks*, Annales Mathematicae et Informaticae, Accepted manuscript doi: 10.33039/ami.2018.09.002

[200] G. Kovásznai, **Cs. Biró**, B. Erdélyi *Generating Optimal Scheduling for Wireless Sensor Networks by Using Optimization Modulo Theories Solvers*, CEUR Workshop Proceedings Vol-1889, 15th International Workshop on Satisfiability Modulo Theories - SMT 2017, pp. 15-27, 2017.

[201] T. Radványi, **Cs. Biró**, S. Király , P. Szigetváry, P. Takács *Survey of attacking and defending in the RFID system*, Annales Mathematicae et Informaticae 44, 151-164, 2015.

[202] **Cs. Biró**, G. Kovásznai, A. Biere, G. Kusper, G. Geda *Cube-and-Conquer approach for SAT solving on grids*, Annales Mathematicae et Informaticae 42 pp. 9-21, 2013.

Conference Papers and Talks

[203] **Cs. Biró**, G. Kusper *BaW 1.0 - A Problem Specific SAT Solver for Effective Strong Connectivity Testing in Sparse Directed Graphs*, IEEE 18th International Symposium on Computational Intelligence and Informatics (CINTI 2018), pp. 160-165, 2018.

[204] G. Kovásznai, **Cs. Biró**, B. Erdélyi *Puli - A Problem-Specific OMT Solver*, 16th International Workshop on Satisfiability Modulo Theories - SMT 2018, Paper: 362, 10 p., 2018.

[205] G. Kusper, **Cs. Biró**, Gy. B. Iszály *SAT solving by CSFLOC, the next generation of full-length clause counting algorithms*, Future IoT Technologies (Future IoT), 2018 IEEE International Conference, pp.1–9, 2018.

[206] G. Kovásznai, B. Erdélyi, **Cs. Biró** *Investigations of graph properties in terms of wireless sensor network optimization*, 2018 IEEE International Conference on Future IoT Technologies, Future IoT 2018, IEEE, pp. 1-8, 2018.

[207] G. Kusper and **Cs. Biró** *Solving SAT by an Iterative Version of the Inclusion-Exclusion Principle*, SYNASC 2015, DOI: 10.1109/SYNASC.2015.38, IEEE Computer Society Press, ISBN 978-1-5090-0461-4, 189–190, 2015.

[208] **Cs. Biró** *Botond - a Simulation and Optimization Framework for Wireless Sensor Networks*, 1st International Conference on Future RFID Technologies and host the Workshop on Smart Applications for Smart Cities Eger, Hungary 6-7 November, conference talk, 2014.

[209] **Cs. Biró**, G. Kusper, T. Radványi, S. Király, P. Szigetváry, P. Takács *SAT Representation of Randomly Deployed Wireless Sensor Networks*, Proceedings of the 9th International Conference on Applied Informatics, pp. 101–111, 2014.

[210] **Cs. Biró** and G. Kusper and T. Tajti *How to generate weakly nondecisive SAT instances* 2013 IEEE 11th International Symposium on Intelligent Systems and Informatics (SISY), pp. 265–269, 2013.

[1]ADATLAP
a doktori értekezés nyilvánosságra hozatalához

I. A doktori értekezés adatai
A szerző neve: Biró Csaba
MTMT-azonosító: 10036956
A doktori értekezés címe és alcíme: Formal Methods for Modelling Wireless Sensor Networks
DOI-azonosító[2]:10.15476/ELTE.2019.163
A doktori iskola neve: ELTE Informatika Doktori Iskola
A doktori iskolán belüli doktori program neve: Az Informatika alapjai és módszertana
A témavezető neve és tudományos fokozata: Dr. Kozsik Tamás PhD, Dr. Kusper Gábor PhD
A témavezető munkahelye: ELTE, EKE

II. Nyilatkozatok
1. A doktori értekezés szerzőjeként[3]

a) hozzájárulok, hogy a doktori fokozat megszerzését követően a doktori értekezésem és a tézisek nyilvánosságra kerüljenek az ELTE Digitális Intézményi Tudástárban. Felhatalmazom az Informatika Doktori Iskola hivatalának ügyintézőjét, Kulcsár Adinát, hogy az értekezést és a téziseket feltöltse az ELTE Digitális Intézményi Tudástárba, és ennek során kitöltse a feltöltéshez szükséges nyilatkozatokat.

b) kérem, hogy a mellékelt kérelemben részletezett szabadalmi, illetőleg oltalmi bejelentés közzétételéig a doktori értekezést ne bocsássák nyilvánosságra az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban;[4]

c) kérem, hogy a nemzetbiztonsági okból minősített adatot tartalmazó doktori értekezést a minősítés (*dátum*)-ig tartó időtartama alatt ne bocsássák nyilvánosságra az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban;[5]

d) kérem, hogy a mű kiadására vonatkozó mellékelt kiadó szerződésre tekintettel a doktori értekezést a könyv megjelenéséig ne bocsássák nyilvánosságra az Egyetemi Könyvtárban, és az ELTE Digitális Intézményi Tudástárban csak a könyv bibliográfiai adatait tegyék közzé. Ha a könyv a fokozatszerzést követőn egy évig nem jelenik meg, hozzájárulok, hogy a doktori értekezésem és a tézisek nyilvánosságra kerüljenek az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástárban.[6]

2. A doktori értekezés szerzőjeként kijelentem, hogy

a) az ELTE Digitális Intézményi Tudástárba feltöltendő doktori értekezés és a tézisek saját eredeti, önálló szellemi munkám és legjobb tudomásom szerint nem sértem vele senki szerzői jogait;

b) a doktori értekezés és a tézisek nyomtatott változatai és az elektronikus adathordozón benyújtott tartalmak (szöveg és ábrák) mindenben megegyeznek.

3. A doktori értekezés szerzőjeként hozzájárulok a doktori értekezés és a tézisek szövegének plágiumkereső adatbázisba helyezéséhez és plágiumellenőrző vizsgálatok lefuttatásához.

Kelt: Eger, 2019. 06. 11.

a doktori értekezés szerzőjének aláírása

---

[1] Beiktatta az Egyetemi Doktori Szabályzat módosításáról szóló CXXXIX/2014. (VI. 30.) Szen. sz. határozat. Hatályos: 2014. VII.1. napjától.
[2] A kari hivatal ügyintézője tölti ki.
[3] A megfelelő szöveg aláhúzandó.
[4] A doktori értekezés benyújtásával egyidejűleg be kell adni a tudományági doktori tanácshoz a szabadalmi, illetőleg oltalmi bejelentést tanúsító okiratot és a nyilvánosságra hozatal elhalasztása iránti kérelmet.
[5] A doktori értekezés benyújtásával egyidejűleg be kell nyújtani a minősített adatra vonatkozó közokiratot.
[6] A doktori értekezés benyújtásával egyidejűleg be kell nyújtani a mű kiadásáról szóló kiadói szerződést.